Hardware-beschleunigte Bildmerkmale mit Subpixel-Genauigkeit zur SLAM Lokalisierung und Objekterkennung

Von der
Carl-Friedrich-Gauß-Fakultät
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines
Doktoringenieurs (Dr.-Ing.)

genehmigte Dissertation

von
Sören Michalik
geboren am 23.06.1985
in Goslar

Eingereicht am:	14.09.2018
Disputation am:	16.11.2018
1. Referentin/Referent:	Prof. Dr.-Ing. Mladen Berekovic
2. Referentin/Referent:	Prof. Dr. rer. nat. Jochen Steil

2018

Abstract

Die Navigation von autonomen Systemen wird durch den Fortschritt der Technik und durch die steigenden Anforderungen der Anwendungen immer komplexer. Eines der wichtigsten offenen Probleme ist die Genauigkeit und die Robustheit der merkmalsbasierten SLAM-Lokalisierung für Anwendungen im dreidimensionalen Raum, da hier durch die Komplexität der Bildverarbeitungsalgorithmen und die Kombination aus vielen einzelnen Komponenten nur eine begrenzte Echtzeitfähigkeit erreicht wird und dadurch der Fehler der Lokalisierung und Kartenerstellung steigt.
Auch der Drift der merkmalsbasierten visuellen Odometrie ist ein nicht zu vernachlässigendes Problem aktueller SLAM-Navigationssysteme mit großflächigen Kartendaten oder Systemen zur Langzeit-Operation von autonomen Systemen.
In dieser Arbeit werden Methoden zur Optimierung der Merkmalserkennung mit Subpixel-genauer Bestimmung der Merkmalsposition für merkmalsbasiserte 6-DoF SLAM Verfahren untersucht. Zusätzlich wird eine Erweiterung des Merkmalsdeskriptors mit Farbinformationen und einer Subpixel-genauen Rotation des Deskriptor-Patterns betrachtet.
Aus den Ergebnissen der Untersuchung wird das Subpixel-accurate Oriented AGAST and Rotated BRIEF (SOARB) Verfahren zur Merkmalserkennung entwickelt, dass trotz der effizienten und Ressourcen-optimierten Implementierung eine Verbesserung der Lokalisierung und Kartenerstellung in Relation zu anderen vergleichbaren Verfahren erreicht.
Durch den Einsatz eines PCIe FPGA-Beschleunigers und der Xilinx SDAccel HW-SW-Codesign Umgebung mit OpenCL Unterstützung wird eine FPGA-basierte Version des SOARB Algorithmus zur Anbindung an SLAM-Systeme gezeigt.
Bei der Hardware-Implementierung wird eine Pipeline-Verarbeitung mit hohem Datendurchsatz und parallelen Einheiten zur Berechnung verwendet. Zur schnelleren Verarbeitung wird die Subpixel-Interpolation sowie eine bilineare Interpolation in Festkomma-Arithmetik durchgeführt und die Winkelberechnung mit einem CORDIC-Verfahren implementiert.
Die FPGA-Implementierung des SOARB-Verfahrens erreicht dabei Bildraten von 41 Bildern/s. Sie ist damit um Faktor 2,6x schneller als die schnellste getestete GPU-basierte Implementierung der OpenCV-Bibliothek mit Sub-

pixel-genauer Bestimmung der Merkmalsposition.
Durch eine geringe Leistungsaufnahme von 13,7W der FPGA-Komponente kann die Leistungseffizienz (Bilder/s pro Watt) des Gesamtsystems im Vergleich zu einer ebenfalls erstellten SOARB GPU-Referenzimplementierung um den Faktor 1,28x gesteigert werden.
Der SOARB-Algorithmus wird zur Evaluation in das RTAB-Map SLAM System integriert und erreicht in Tests mit Bildaufnahme-Sequenzen aus dem Straßenverkehr eine Verbesserung des Translations- und Rotationsfehlers von durchschnittlich 22% und 19% im Vergleich zu dem häufig genutzten ORB-Verfahren. Die maximale Verbesserung des Root Mean Square Errors (RMSE) liegt bei 50% für die Translation und 40% für die Rotation.
Durch einen Deskriptor mit Farbinformationen kann das SOARB-RGB Verfahren in der Evaluation mit dem Oxford Datensatz zur Bewertung von affinen kovarianten Merkmalen ein sehr gutes Inlier-Verhältnis von 99,2% über die ersten drei Bildvergleiche aller Datensätze erzielen.
Die Ergebnisse zeigen, dass durch das vorgestellte SOARB-Verfahren und einen FPGA-Beschleuniger die Lokalisierung und Kartenerstellung des RTAB-Map SLAM-Systems in ihrer Genauigkeit und Echtzeit-Fähigkeit verbessert werden kann.

English Version

The navigation of autonomous systems is becoming more and more complex due to advances in technology and the increasing demands of applications. One of the most critical open issues is the accuracy and robustness of feature-based SLAM localization for three-dimensional SLAM applications, where only limited real-time capability can be achieved due to the complexity of the image processing algorithms and the interaction of many individual components increasing the localization and mapping error.
Also, the drift of feature-based visual odometry is a not negligible problem of current SLAM navigation systems with large area maps or systems for long-term operation of autonomous systems.
In this work the optimization of feature detection with subpixel-accurate features points for feature-based 6-DoF SLAM methods is investigated. In addition, an extension of the feature descriptor with color information and sub-pixel accurate rotation of the descriptor pattern is evaluated.
This work develops a Subpixel-accurate Oriented AGAST and Rotated BRIEF (SOARB) feature extraction that, despite the efficient and resource-optimized implementation, improves localization and mapping compared to other comparable algorithms.
Using a PCIe FPGA accelerator and the Xilinx SDAccel HW-SW Codesign environment with OpenCL support an FPGA-based version of the SOARB algorithm for interfacing to SLAM systems is demonstrated.
The hardware implementation uses high-throughput pipeline processing and

parallel units for computation. For faster processing, the subpixel interpolation and a bilinear interpolation is performed in fixed-point arithmetic and the angle calculation is implemented using a CORDIC method.
The FPGA implementation of the SOARB algorithm achieves frame rates of 41 frames/s. Thus, it is a factor of 2.6 times faster than the fastest of the tested GPU-based OpenCV implementation with subpixel-accurate feature positions.
With a low power consumption of 13.7W of the FPGA component, the overall system power efficiency (fps per watt) can be increased by a factor of 1.28x compared to an implemented SOARB-GPU reference implementation.
For evaluation the SOARB algorithm is integrated into the RTAB Map SLAM system. It achieves an average of 22% and 19% improvement in translational and rotational errors compared to the commonly used ORB feature extraction in tests with dataset sequences for autonomous driving. The maximum improvement in root mean square error (RMSE) is 50% for translation and 40% for rotation.
To analyze the impact of descriptor with color information, the SOARB-RGB method ist evaluated using the Oxford dataset for affine covariant features. The SOARB-RGB achieves a very good inlier-ratio of 99.2% over the first three dataset image of all datasets.
The results show that the presented SOARB method and FPGA accelerator can improve the accuracy and real-time capability of location and mapping the RTAB Map SLAM significantly.

Glossar

AGAST	Adaptive and Generic Accelerated Segment Test
ASIC	Application-Specific Integrated Circuit
BRIEF	Binary Robust Independent Elementary Features
BRISK	Binary Robust Invariant Scalable Keypoints
CPU	Central Processing Unit
DDR	Double Data Rate Memory
DoF	Degrees of Freedom
DoG	Difference of Gaussian
FAST	Features from Accelerated Segment Test
FPGA	Field Programmable Gate Array
FREAK	Fast Retina Keypoint
GPU	Graphics Processing Unit
HLS	High Level Synthese
LUT	Look-up Table
ORB	Oriented FAST and Rotated BRIEF
RAM	Random Access Memory
RGB	Rot, Grün, Blau (Farbkanäle)
SIFT	Scale-Invariant Feature Transform
SLAM	Simultaneous Localization and Mapping
SoC	System on Chip
SURF	Speeded-Up Robust Features
SRCC	Sparse Retina Census Correlation

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Navigation von mobilen Robotern und autonomen Fahrzeugen ist eines der größten Probleme in der Robotik der heutigen Zeit. Autonome Systeme werden durch den Fortschritt der Technik immer komplexer und sollen sich in der Umgebung von Menschen eigenständig und sicher bewegen können. Anwendungsgebiete sind unter anderem mobile Service Roboter im Haushalt und in der Altenpflege, selbst-fahrende Fahrzeuge, industrielle mobile Roboter, Logistik-Systeme sowie mobile Sicherheitssysteme.

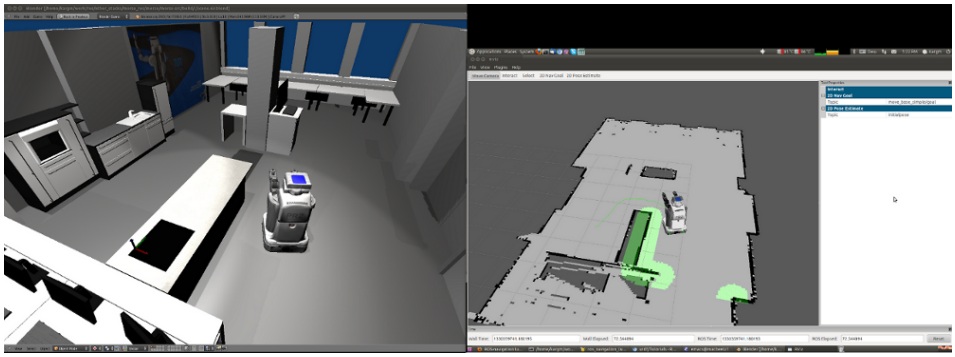
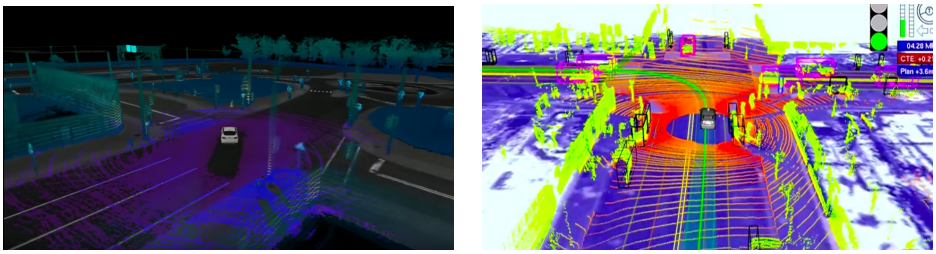


Abbildung 1.1: 2D Roboter Navigation in ROS [Ope18]

Zur Navigation in unbekannten Umgebungen wird Simultaneous Localization and Mapping (SLAM) verwendet, um aus Sensordaten eine Karte der Umgebung zu erstellen und gleichzeitig das Roboter-System darin zu lokalisieren. Ist die genaue Position gefunden, kann der Roboter durch Wegplanung zu gewünschten Zielen auf der Karte navigiert werden.
Die Schwierigkeit der SLAM-Navigation liegt darin, die Kartographierung und Lokalisierung gleichzeitig auszuführen.
Man unterscheidet bei den SLAM Verfahren zwischen 2D-SLAM und vollständigem 3D(6D)-SLAM. Bei den zweidimensionalen Verfahren erfolgt die Lokalisierung und Kartographierung nur in einer Ebene mittels 2D-Laser-



(a) (b)

Abbildung 1.2: 3D SLAM im Automobilbereich [CB18] [Gui18]

scannern oder 2D-Projektionen von Tiefenkameras. Diese Verfahren bieten sich daher für die Navigation oder Kartenerstellung in Anwendungen mit nur einer Ebene wie z.B. in Indoor-Bereichen oder Bürogebäuden an.
Bei den aufwändigeren dreidimensionalen bzw. 6DoF (Degree of Freedom) Verfahren wird durch Verwendung von Stereo- oder RGBD-Tiefenkameras die Kameraposition in allen 6 Freiheitsgraden bestimmt und verfolgt, daher eignen sich diese Algorithmen für Outdoor-Anwendungen oder für komplexe Applikationen mit unebenen Umgebungen.
Merkmalsbasierte 6DoF-SLAM Algorithmen nutzen dabei Bildmerkmale aus aufeinanderfolgenden Bildern, um für die Kartographierung Landmarken aus sog. Merkmals-Bündeln zu erstellen. Daraus lassen sich dann Kartendaten generieren, die Kameraposition verfolgen und in der Navigation bereits erkannte Kartenabschnitte wiedererkennen.
Sollen lediglich die Kamerabewegungen verfolgt werden, genügen zur Bewegungsverfolgung ein Merkmals-Detektor in Kombination mit einem Tracking-Verfahren [LK+81] [TK91]. Sollen hingegen auch Kartenbereiche wiedererkannt und miteinander verknüpft werden können, wird zusätzlich ein Merkmals-Deskriptor benötigt, um einzelne Merkmale miteinander vergleichen zu können. In dieser Arbeit sollen die SLAM-Verfahren mit Deskriptor-basierter Merkmalsverfolgung betrachtet werden.
Aktuelle bildbasierte SLAM Verfahren bieten trotz stetiger Verbesserung der verwendeten Prozessortechnik nur eine begrenzte Echtzeit-Fähigkeit aufgrund der Komplexität der einzelnen Bildverarbeitungskomponenten. Soll ein SLAM-System Echtzeit-fähig sein, wird oft entweder die Anzahl der verwendeten Merkmale reduziert oder die Qualität der Merkmale vereinfacht, was wiederum negative Auswirkungen auf die Genauigkeit der Kartenerstellung hat.
Auch der Drift aufgrund von Abweichungen bei der Berechnung der Kamerabewegung aus Bildinformationen, der sog. visuellen Odometrie, ist ein nicht zu vernachlässigendes Problem aktueller SLAM-Navigationssysteme. Dies ist eine besondere Herausforderung für Anwendungen mit großflächigen Kartendaten oder Anwendungen mit Langzeit-Operation von autono-

men Systemen. Der Drift der visuellen Odometrie lässt sich dabei nur durch eine gute Qualität der Merkmalserkennung reduzieren.
Merkmalsbasierte SLAM-Systeme nutzten derzeit monochrome Bilder, um die Komplexität der Verarbeitung zu reduzieren. Sind Farb-Kameras auf einem Roboter-System vorhanden, geht der zusätzliche Informationsgehalt aus aufgenommenen Farbbildern dabei durch eine Konvertierung in Graustufen verloren.
Die hier gezeigten Probleme bieten einige Ansatzpunkte zur Optimierung von Merkmalserkennungen für SLAM-Systeme. So kann durch eine Hardware-gestützte Implementierung und durch eine direkte Anpassung der Algorithmen auf SLAM-Systeme idealerweise die Echtzeit-Fähigkeit und die Qualität der Merkmale erhöht werden. Zusätzlich könnten durch die Leistungsreserven einer Hardware-gestützten Implementierung auch eine komplexere Verarbeitung mit Farbbildern integriert werden, um eine verbesserte Zuordnung von Merkmalen zu ermöglichen.

1.1 Struktur der Arbeit, Zielstellung und eigene Beiträge

In dieser Arbeit sollen die für merkmalsbasierte 6-DoF SLAM verwendeten Merkmals-Detektions- und Deskriptionsverfahren untersucht werden und durch die Implementierung auf Systemen mit FPGA- und GPU-Hardware-Beschleunigern in ihrer Genauigkeit und Echtzeit-Fähigkeit verbessert werden.
Dazu werden zunächst in Kapitel 2 die Grundbegriffe der SLAM-Lokalsierung und Merkmalserkennung sowie einzelne Verfahren der Merkmalsedetektion und Merkmalsdeskription vorgestellt.
Kapitel 3 gibt eine Übersicht über andere verwandte Arbeiten zur Beschleunigung von Merkmalserkennungen und den Einsatz von Beschleunigern in SLAM-Anwendungen.
Möglichkeiten zur Optimierung von Merkmalserkennungs-Verfahren zum Einsatz in SLAM-Systemen werden in Kapitel 4 vorgestellt und analysiert.
In den Kapiteln 5, 6 und 7 wird, als Kernpunkt dieser Arbeit, ein neuartiges Verfahren zur Merkmalserkennung mit der Bezeichnung Subpixel-accurate Oriented AGAST and Rotated BRIEF (SOARB) entworfen und in ein SLAM-System mit FPGA-Beschleuniger integriert.
Der SOARB Algorithmus wird in Kapitel 8 mit verschiedenen Datensätzen zur Detektor-/Deskriptor-Evaluation analysiert und ebenfalls in Kombination mit einem merkmalsbasierten SLAM-System und Bildaufnahmen aus dem Straßenverkehr getestet.
Im abschließenden Kapitel 9 werden die Ergebnisse der Evaluation der Hardware-basierten Merkmalserkennung für SLAM-Systeme zusammengefasst und reflektiert.

Im Verlauf der Arbeit werden folgende Beiträge erarbeitet:

- Analyse und Auswahl von Verfahren zur Subpixel-genauen Merkmalserkennung für eine Implementierung in integrierten Schaltkreisen
- Entwurf eines auf SLAM-Anwendungen optimiertes Merkmalserkennungsverfahren
- Hardware-Beschleunigung der Merkmalserkennung mit einem FPGA-Beschleuniger und Integration in ein SLAM-System
- Erweiterung der Merkmalsbeschreibung mit mehreren Farbkanälen zur Deskriptorgenerierung für SLAM-Anwendungen
- Evaluation des Hardware-beschleunigten SLAM-Systems mit Hilfe von Datensätzen zur Bewertung von Merkmalserkennungen und Datensätzen zur Bewertung von SLAM-Systemen
- Verbesserung des Translationsfehlers von durchschnittlich 22% und bis zu 50% in einzelnen Datensätzen

Dazu werden in dieser Dissertation Verfahren zur Subpixel-genauen Interpolation der Merkmalsposition und zur Subpixel-genauen Intensitäts-Interpolation in der Deskriptorberechnung betrachtet. Zusätzlich wird der Einsatz von mehreren Farbkanälen zur Deskriptorgenerierung für SLAM-Anwendungen diskutiert.
Der Fokus dieser Arbeit liegt neben der Optimierung der Merkmalsextraktion auf der Entwicklung eines SLAM-Systems mit einem FPGA Hardware-Beschleuniger, da durch kürzere Berechnungszeiten die Ergebnisse und Stabilität der Lokalisierung verbessert werden können.
Ein wichtiger Aspekt für die Navigation mobiler Systeme ist ebenfalls die Leistungsaufnahme, da hier meist nur begrenzte Kapazitäten verfügbar sind und die Akku-Laufzeit stark von den eingesetzten Bildverarbeitungskomponenten abhängt. Die geringere Stromaufnahme eines FPGA-Systems im Vergleich zu GPU-Systemen ermöglicht hier Vorteile beim Einsatz in mobilen Systemen, daher soll in dieser Arbeit auch der Energieverbrauch der Implementierungen untersucht werden.
Zur Umsetzung und Entwicklung der Hardware-Komponenten wird in dieser Arbeit das Xilinx SDAccel Hardware-Software-Codesign Framework verwendet, um einen Beschleuniger in SLAM-Systeme zu integrieren. Hierdurch lassen sich Nachteile der FPGA-Entwicklung umgehen und die komplexen Speicheranforderungen von Merkmalserkennungsverfahren erfüllen.

Kapitel 2

Grundlagen

2.1 Merkmalsbasierte SLAM-Systeme

Merkmalsbasierte SLAM-Systeme nutzen Bildmerkmale zur Bestimmung der Position eines mobilen Systems auf einer Karte. Dazu werden die Beobachtungen von wiedererkennbaren Bildbereichen der Umgebung (sog. Landmarken) genutzt, um die aktuelle Position zu schätzen. Weil nur relative Messungen aufgrund der Beobachtung der Landmarken möglich sind, enthält die Positionsbestimmung eine Unsicherheit.

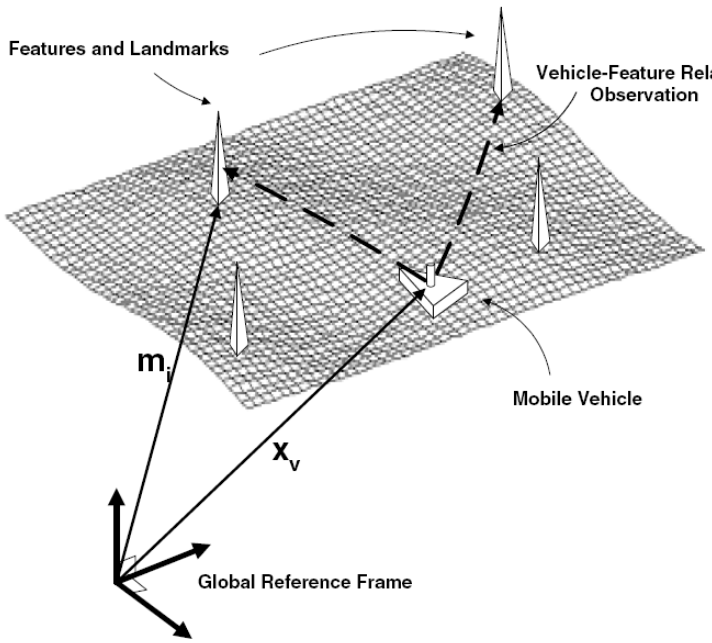


Abbildung 2.1: Merkmalsbasiertes SLAM-System mit Landmarken [YMOFE17]

In Bild 2.1 ist ein merkmalsbasiertes SLAM System mit Landmarken dargestellt. Um die Position der Landmarken im Raum zu erhalten, ist neben den Merkmalskoordinaten im Bild auch die Tiefe zu den Merkmalen bzw. der Abstand der Landmarken wichtig. Die Hauptbestandteile eines merkmalsbasierten SLAM-Systems sind in Abbildung 2.2 gezeigt.
Für die visuelle Odometrie werden Bildmerkmale zunächst detektiert und mit einem Deskriptor beschrieben. Im nächsten Schritt werden die Bildtiefen zu allen erkannten Merkmalen bestimmt.

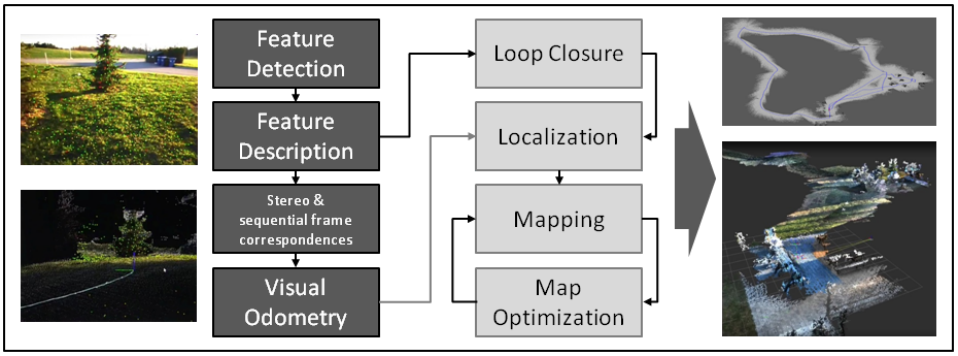
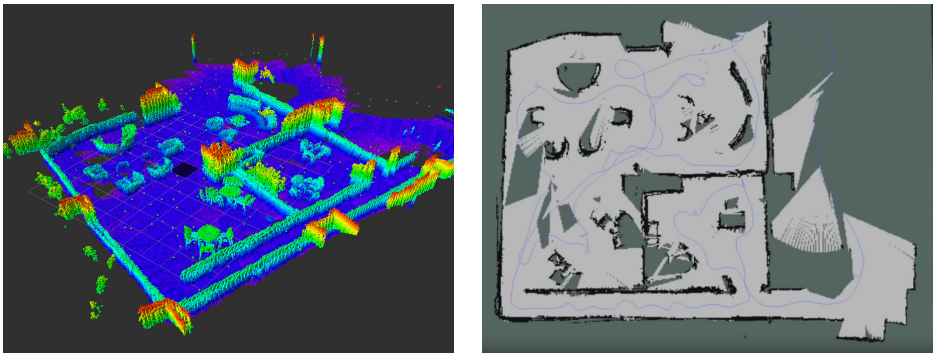


Abbildung 2.2: Aufbau eines merkmalsbasierten SLAM-Systems

Zur Kartenerstellung werden mit diesen Informationen die relativen Positionen der Landmarken in einer Karte gespeichert. Damit kann durch Deskriptor-Vergleiche die Lokalisierung in der Karte sowie die Wiedererkennung von bereits aufgedeckten Kartenbereichen erfolgen.
Die Wiedererkennung und Verbindung von bereits aufgedeckten Kartenabschnitten wird auch als Loop Closure bezeichnet. Damit lassen sich die durch Odometrie-Fehler verursachten Abweichungen ausgleichen und somit die Kartenerstellung verbessern.
Zusätzlich wird in den meisten SLAM-Verfahren die Karte zur Laufzeit optimiert, um die gespeicherten Daten zu reduzieren und die Qualität der Landmarken stetig zu verbessern.
In den nächsten Kapiteln sollen die merkmalsbasierten SLAM Systeme RTAB-Map (2.1.1) und ORB-SLAM2 (2.1.2) sowie die verwendeten Algorithmen zur Merkmalserkennung näher beschrieben und analysiert werden.

2.1.1 RTAB-Map

Der Real-Time Appearance-Based Mapping (RTAB-Map) Algorithmus [LM11] [LM13] ist ein merkmalsbasiertes 6DoF SLAM Verfahren mit Appearance-Based Loop-Closure Detektion und Graphen-basiertem Speicher-Management zur Einhaltung von Echtzeit-Anforderungen. RTAB-Map erstellt dazu aus 3D-Sensordaten von RGBD-Kameras oder Stereo-Kameras zu jedem Eingangsbild eine Signatur aus Merkmalen und Tiefendaten. Diese kann dann als Teil eines Graphen gespeichert werden und mit Signaturen aus vorher besuchten Kartenbereichen verglichen werden, um globale Loop-Closures zu ermöglichen. Bei Bedarf können auch 2D-Laserscanner-Daten integriert werden.



(a) (b)

Abbildung 2.3: RTAB-Map SLAM: (a) 3D-Karte [LM14] (b) 2D-Projektion der Karte [LM14]

RTAB-Map benutzt ein Graphen-basiertes Speicher-Management-System und ein Bayes'sches Modell, um Echtzeit-Anforderungen für die Erstellung großer Karten im Langzeitbetrieb zu erfüllen.
Der Algorithmus teilt den verwendeten Speicher dazu in Working Memory (WM), Short-Term Memory (STM) und Long-Term Memory (LTM). Wenn genügend Merkmale von guter Qualität in den Sensordaten gefunden wurden, wird eine Signatur im Short-Term Memory erstellt. Der STM wird verwendet, um Ähnlichkeiten zwischen aufeinanderfolgenden Bildern zu beobachten und die Signaturen nach einer bestimmten Anzahl von Bildern, abhängig von der Bewegungsgeschwindigkeit des Roboters und der eingestellten Aktualisierungsrate, in den Working Memory zu übertragen. Dabei wird der STM nicht für Loop-Closures benutzt, um keine Loop-Closures mit gerade aufgenommen Signaturen zu ermöglichen.
Der Working Memory enthält dann die Signaturen, die kürzlich aufgenommen wurden oder sich oft wiederholen, um die Bearbeitungszeit pro aufgenommenem Bild zu beschränken. Die Größe des Working Memory kann so

gewählt werden, dass ein vorgegebenes Kriterium für die Echtzeit-Verarbeitung nicht verletzt wird.

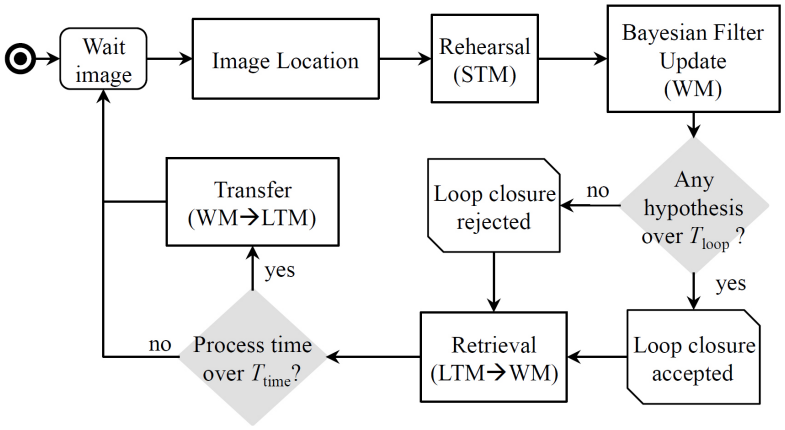


Abbildung 2.4: Ablauf-Diagramm des RTAB-Map SLAM [LM11]

In den Long-Term Memory werden die Signaturen kopiert, die zu dem aktuellen Zeitpunkt nicht so häufig verwendet werden oder wenn ein Loop-Closure zwischen dem aktuellen Frame und dem Working Memory gefunden wurde. Die Autoren vergleichen dieses Verhalten mit dem menschlichen Langzeit-Gedächtnis, das mehr Informationen zu häufig besuchten Orten als zu weniger besuchten Orten speichert.

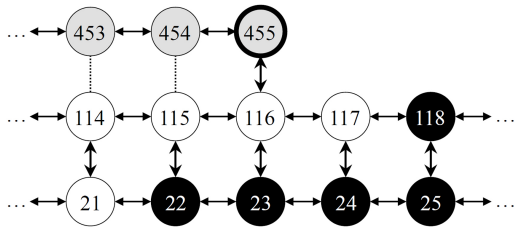
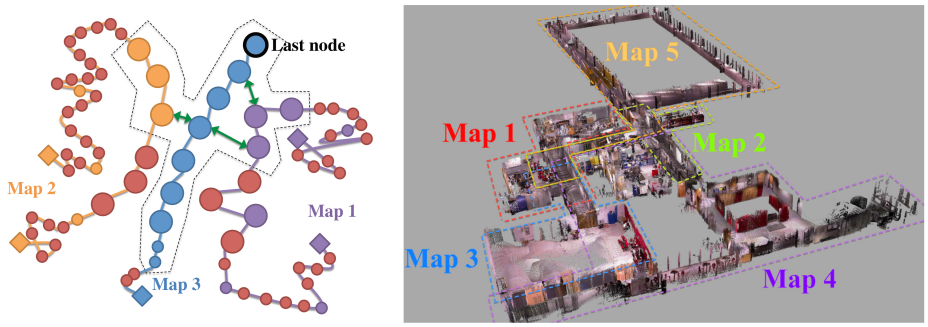


Abbildung 2.5: RTAB-Map Graphen-basiertes Speichermanagement [LM13]

Die visuelle Odometrie ist in RTAB-Map als separates Modul implementiert und kann über Parameter auf verschiedene Odometrie-Verfahren konfiguriert werden. So können entweder die internen merkmalsbasierten Ansätze oder zusätzliche bestehende Verfahren wie VISO2 [GZS11], FOVIS [HBH+17], DVO [KSC13] oder die Odometrieberechnung von ORB-SLAM2 [MAT17] genutzt werden.
RTAB-Map bietet ebenfalls eine gute Unterstützung aktueller Bildverarbeitungsverfahren zur Erkennung von Bildmerkmalen aus der OpenCV-



(a) (b)

Abbildung 2.6: (a) Multi-session Map-Graph (b) Multi-session Map [LM14]

Bibliothek. Durch das konfigurierbare Speichermanagement-System kann die Kartenerstellung sehr gut an die verwendeten Verfahren zur Merkmals-Detektion angepasst werden.
RTAB-Map ist als Open-Source Software verfügbar und bietet ebenfalls eine sehr gute Unterstützung für das ROS-System [QCG+09]. Die Integration in ROS ist über Wrapper-Nodes realisiert, um zusätzlich auch externe Komponenten wie z.B. eine externe visuelle Odometrie verwenden zu können.

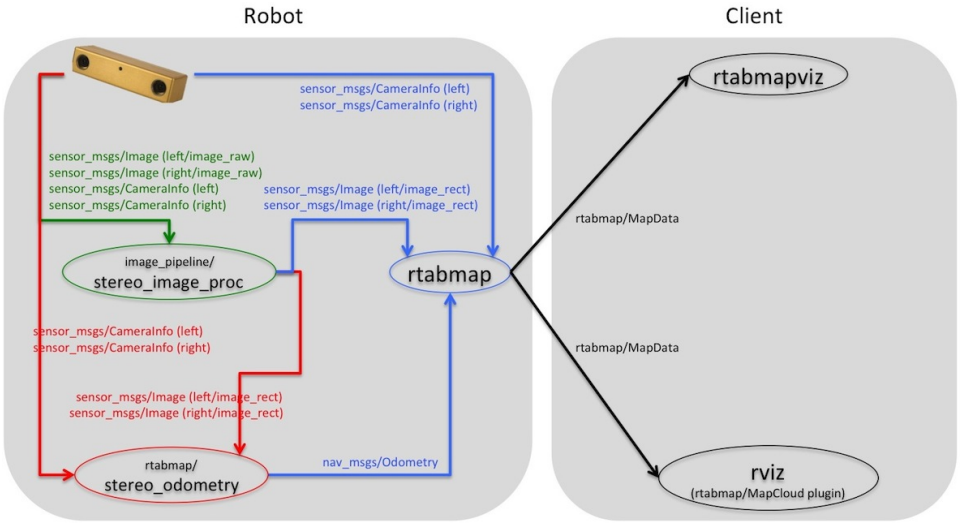
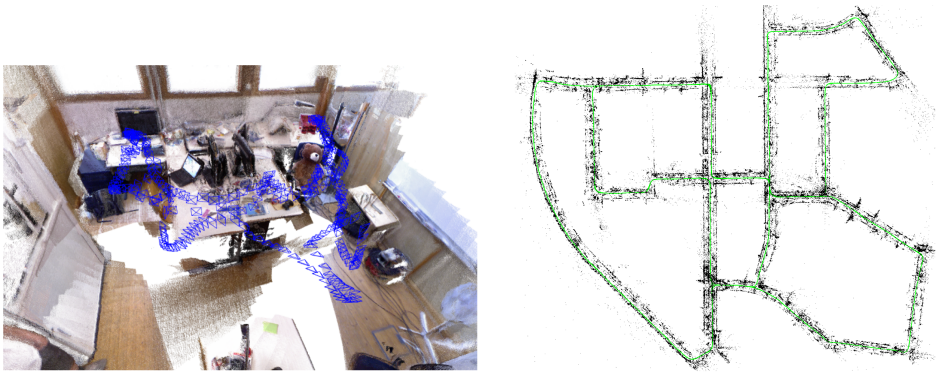


Abbildung 2.7: RTAB-Map ROS-Konfiguration mit Stereo-Kamera [Lab18]

Abbildung 2.7 zeigt eine RTAB-Map ROS-Konfiguration mit Stereo-Kamera und die ROS-Topics zur Kommunikation zwischen den Nodes. RTAB-Map bietet darüber hinaus eine Option zur Erstellung von 3D-Punktwolken aus den gesammelten Navigations- und Sensordaten sowie die Generierung von OctoMap-Daten.

2.1.2 ORB-SLAM2

ORB-SLAM2 [MAT17] ist ein merkmalsbasiertes SLAM-System für monokulare, Stereo und RGBD-Kameras mit Unterstützung für Karten-Wiederverwendung, Loop-Closures und Re-Lokalisierung. Das System ist eine Weiterentwicklung des monokularen ORB-SLAM Algorithmus und nutzt ebenfalls eine angepasste ORB-Merkmalserkennung zur Verfolgung von Kamerabewegungen, Kartenerstellung und Wiedererkennung.



(a) (b)

Abbildung 2.8: (a) ORB-SLAM2 - generierte 3D Punktwolke [MAT17] (b) ORB-SLAM2 - Bahnkurve und Rekonstruktion einer Stadt-Umgebung [MAT17]

Der Algorithmus setzt sich aus drei Hauptkomponenten zusammen:

- Die Verfolgung und Lokalisierung der Kamera durch Merkmalsvergleiche mit einer lokalen Karte und Minimierung der Re-Projektionsfehler
- Eine lokale Kartenerstellung und Kartenoptimierung über Bundle Adjustment (BA)
- Loop-Closing zur Korrektur von Drift durch Graphen-Optimierung

Bundle Adjustment ist dabei ein bekanntes Verfahren aus der Photogrammetrie zur verbesserten Rekonstruktion einer 3D-Szene durch Optimierung der Sehstrahlenbündel von Bildaufnahmen aus mehreren Perspektiven. Dazu können neben den Positionen der Punkte im 3D-Raum auch die Koordinaten und Orientierungen der beobachtenden Kameras variiert werden, um den Re-Projektions-Fehler zu minimieren.
Eine detaillierte Übersicht des ORB-SLAM2 Systems ist in Abbildung 2.9 gezeigt.
Zur Verfolgung der Kameraposition über Bildmerkmale und zur Kartographie erstellt ORB-SLAM sog. Keyframes. Keyframes sind dabei Merkmals-

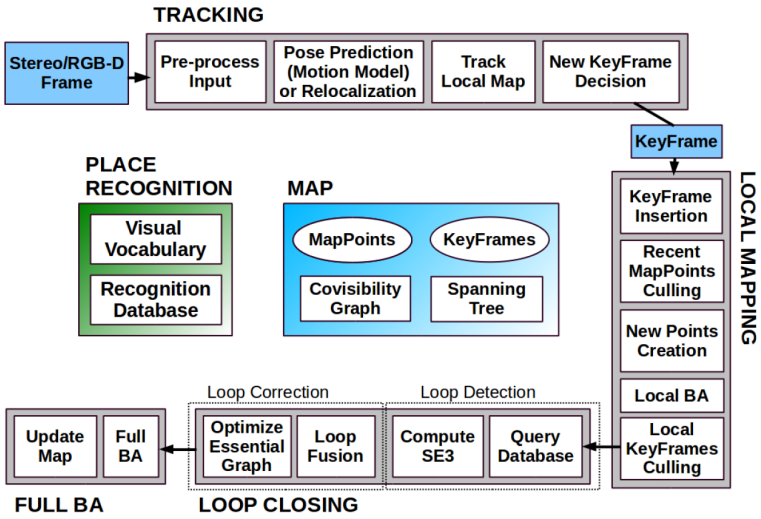


Abbildung 2.9: Systemübersicht des ORB-SLAM2 [MAT17]

Sammlungen mit einer guten Qualität aus Bildaufnahmen zu bestimmten Zeitpunkten.
Zum Start wird zunächst ein initialer Keyframe erstellt. Diese Merkmale werden dann zum Verfolgen der Kameraposition genutzt, solange genügend Merkmals-Übereinstimmungen vorhanden sind. Sobald die Anzahl der zur Verfolgung nutzbaren Punkte unter einen bestimmen Wert fällt, wird ein neuer Keyframe erstellt.

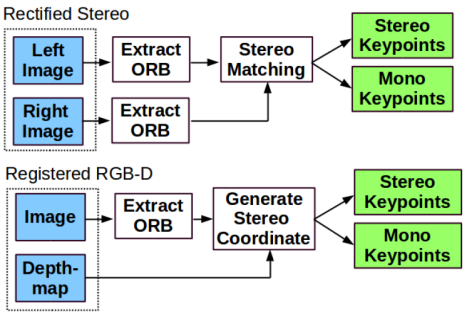


Abbildung 2.10: Eingangsdaten-Vorverarbeitung in ORB-SLAM2 [MAT17]

In [MAT17] wurde ORB-SLAM2 mit verschiedenen Datensätzen evaluiert. Das System zeigt dabei sehr geringe Fehlerraten im Vergleich zu anderen State-of-the-Art Verfahren. In der Arbeit konnte gezeigt werden, dass Bundle Adjustment zur Kamera-Lokalisierung bessere Ergebnisse liefern kann als die rechenaufwändigen direkten oder Iterative Closest Point (ICP) Verfah-

ren. Zusätzlich wurden in [MAT17] die Laufzeiten aller Threads untersucht. Tabelle 2.11 zeigt die gemessenen Laufzeiten der ORB-SLAM2 Komponenten. Die parallelen Threads laufen dabei mit unterschiedlichen Wiederholungsraten, um trotz der hohen Laufzeiten für die Aktualisierung der Karteninformationen die Anforderungen der Lokalisierung bestmöglich zu erfüllen. Die Tracking-Komponente zur Verfolgung von Kamerabewegungen hat dabei in SLAM-Systemen die höchsten Echtzeit-Anforderungen. Hierbei ist zu sehen, dass in der Tracking-Komponente die ORB Extraktion enthalten ist und bei einer Extraktion von 2000 Merkmalen bereits nahezu 50% der Gesamtzeit einnimmt.

Settings	Sequence	V2_02	07	fr3_office
	Dataset	EuRoC	KITTI	TUM
	Sensor	Stereo	Stereo	RGB-D
	Resolution	752×480	1226×370	640×480
	Camera FPS	20Hz	10Hz	30Hz
	ORB Features	1000	2000	1000
Tracking	Stereo Rectification	3.43 ± 1.10	-	-
	ORB Extraction	13.54 ± 4.60	24.83 ± 8.28	11.48 ± 1.84
	Stereo Matching	11.26 ± 6.64	15.51 ± 4.12	0.02 ± 0.00
	Pose Prediction	2.07 ± 1.58	2.36 ± 1.84	2.65 ± 1.28
	Local Map Tracking	10.13 ± 11.40	5.38 ± 3.52	9.78 ± 6.42
	New Keyframe Decision	1.40 ± 1.14	1.91 ± 1.06	1.58 ± 0.92
	Total	41.66 ± 18.90	49.47 ± 12.10	25.58 ± 9.76
Mapping	Keyframe Insertion	10.30 ± 7.50	11.61 ± 3.28	11.36 ± 5.04
	Map Point Culling	0.28 ± 0.20	0.45 ± 0.38	0.25 ± 0.10
	Map Point Creation	40.43 ± 36.10	47.69 ± 29.52	53.99 ± 23.62
	Local BA	137.99 ± 248.18	69.29 ± 61.88	196.67 ± 213.42
	Keyframe Culling	3.80 ± 8.20	0.99 ± 0.92	6.69 ± 8.24
	Total	174.10 ± 278.80	129.52 ± 88.52	267.33 ± 245.10
Loop	Database Query	3.57 ± 5.86	4.13 ± 3.54	2.63 ± 2.26
	SE3 Estimation	0.69 ± 1.82	1.02 ± 3.68	0.66 ± 1.68
	Loop Fusion	21.84	82.70	298.45
	Essential Graph Opt.	73.15	178.31	281.99
	Total	108.59	284.88	598.70
BA	Full BA	349.25	1144.06	1640.96
	Map Update	3.13	11.82	5.62
	Total	396.02	1205.78	1793.02
Loop size (#keyframes)		82	248	225

Abbildung 2.11: Laufzeiten der einzelnen ORB-SLAM2 Threads in Millisekunden [MAT17]

ORB-SLAM2 ist als Open-Source Implementierung verfügbar. Von der ETH Zürich wird zusätzlich ein ROS-Interface bereitgestellt. Das Interface enthält alle wichtigen Komponenten für die SLAM-Navigation, bietet aber im Vergleich zu RTAB-Map keine direkte Erstellung von 3D-Punktwolken aus RGBD-Daten oder eine Generierung von OctoMap-Daten.

2.2 Merkmalserkennung

Bildvergleiche und Bildregistrierung von Bildern aus unterschiedlichen Bildaufnahmen und Blickrichtungen stellen eine große Herausforderung für Bildverarbeitungsanwendungen dar. Die Idee der Merkmalserkennung ist es, einen Bildbereich in kleinere lokale Segmente, sogenannte Bildmerkmale, zu unterteilen, um durch Kombination der Merkmale eine robustere Beschreibung und Vergleichsmöglichkeit von Bildbereichen zu erhalten.

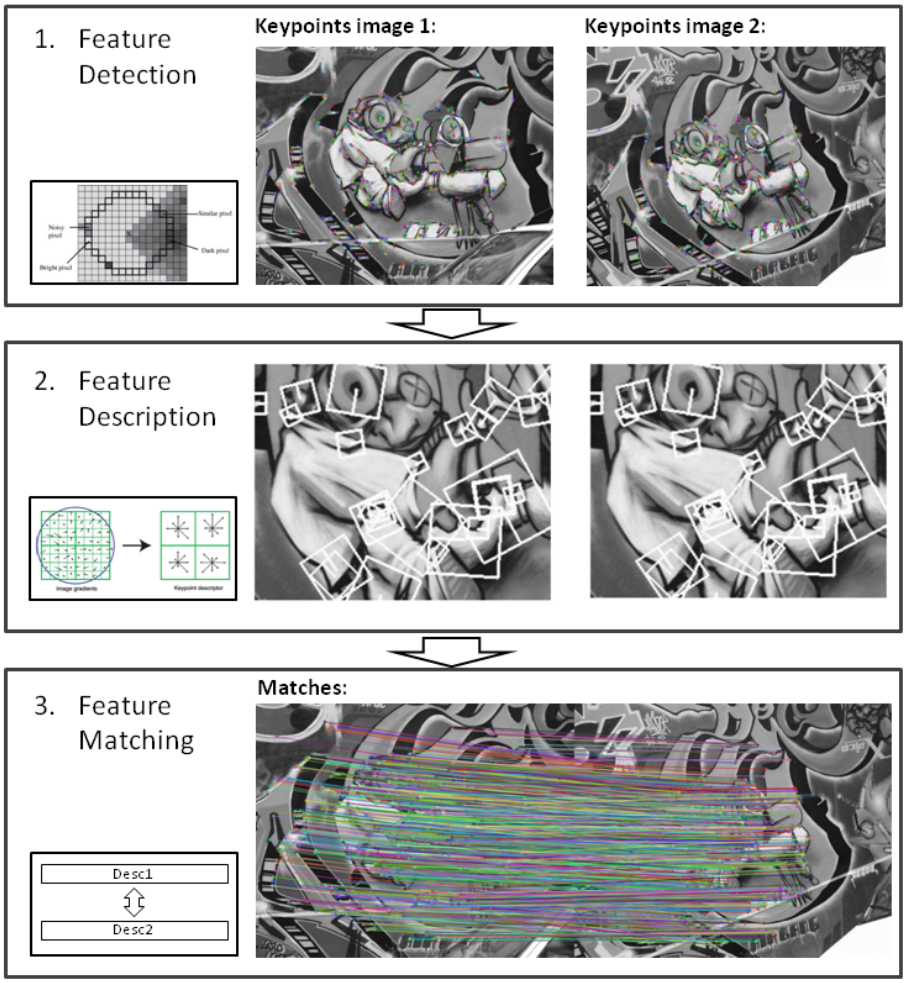


Abbildung 2.12: Ablauf der Merkmalserkennung

Bildmerkmale sind Bildregionen mit wiedererkennbaren Strukturen und Eigenschaften, wie z.B. Bereiche mit großen Gradientenwerten durch Kanten und Ecken. Dabei sind die meisten Merkmalserkennungssysteme auf ein Implementierungs-spezifisches Kantenkriterium optimiert. Einzelne lokale Bildmerkmale können dann durch Beschreibung der Merkmalsumgebung

mit einen Merkmals-Deskriptor und durch Vergleich mit Deskriptoren aus anderen Bilder der gleichen Szene wiedererkannt werden.
Das Vergleichen von Deskriptoren wird auch als Deskriptor-Matching bezeichnet und wird je nach Implementierung des Merkmals-Deskriptors über die Berechnung des euklidischen Abstands oder die Berechnung einer Hamming-Distanz ausgeführt. Für die Merkmals-Detektoren, oft auch als Kantendetektoren bezeichnet, sind dabei Eigenschaften wie Wiederholbarkeit der Erkennung wichtig. Sie sollen, wie auch die Merkmals-Deskriptoren, invariant gegenüber Rotation, Skalierung, Helligkeitsänderungen sowie Bildrauschen sein. Ein Deskriptor soll dabei eine möglichst einzigartige Beschreibung des Merkmals bieten. Aufgrund des lokalen Aufbaus der Merkmalerkennung und der dadurch resultierenden Robustheit gegenüber Bildverzerrungen und Okklusion lassen sich die Verfahren sehr gut in Lokalisierungsanwendungen, SLAM-Navigation und auch der Stereo-Tiefenbild-Berechnung (Sparse Stereo) verwenden. Durch Zusammenfassung mehrerer Merkmale können Bildmerkmale ebenfalls zur Objekterkennung genutzt werden. Der Ablauf einer Merkmalserkennung ist in Abbildung 2.12 gezeigt.

2.2.1 Binäre Deskriptoren und Deskriptoren mit Vektoren von Fließkommazahlen

Viele Merkmals-Deskriptoren wie z.B. SIFT und SURF verwenden Vektoren von Fließkommazahlen zur Beschreibung der Merkmalsregion. Diese bieten sehr gute Erkennungsraten über den gesamten Skalenraum. Die Deskriptoren enthalten einzelne Gradienten als Fließkommazahlen und benötigen daher viel Speicherplatz (256-512Bytes pro Merkmal). Der Vergleich von den Deskriptoren ist dadurch ebenfalls vergleichsweise rechenaufwändig, da das Matching über euklidische Abstände berechnet wird.

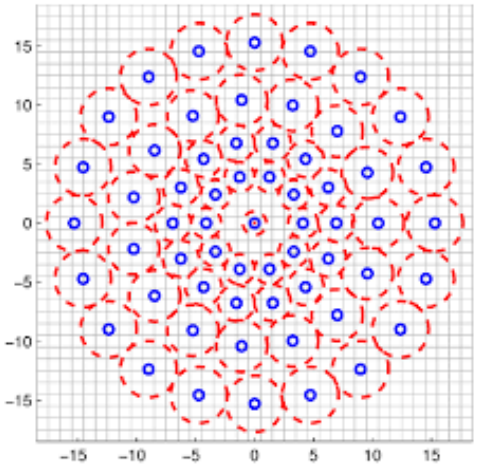


Abbildung 2.13: BRISK Deskriptor Pattern[LCS11]

Um den Rechenaufwand in der Merkmalsbeschreibung und im Matching zu reduzieren, wurden sog. binäre Merkmalsdeskriptoren [CLSF10] vorgestellt. Binäre Merkmalsdeskriptoren verwenden Bitstrings zur Beschreibung der Bildmerkmale. Dabei werden eine große Anzahl von Pixel-Intensitätsvergleichen nach Formel 2.1 durchgeführt. Ein Beispiel eines binären Deskriptor-Patterns ist in Abbildung 2.13 gezeigt.

$$\tau(p;x;y) = \begin{cases} 1, & p(x) \geq p(y) \\ 0, & otherwise \end{cases} \tag{2.1}$$

Das Ergebnis jedes Vergleichs wird als Bit-Wert im Deskriptor übernommen. Die Idee dahinter ist, dass jedes Bit im Deskriptor unabhängig ist und ein Deskriptor-Vergleich damit über leichter zu berechnende binäre Operationen durchgeführt werden kann. Dabei haben die meisten binären Deskriptoren mit Längen von 256-512 Bit einen vergleichsweise geringen Speicherbedarf. Zum Vergleich von zwei binären Deskriptoren kann die Hamming Distanz verwendet werden. Die Hamming-Distanz zählt die unterschiedlichen Bits in zwei Deskriptoren und kann sehr effizient durch eine XOR-Operation ausgeführt werden. Dadurch ist sie deutlich schneller durchzuführen als die Berechnung des euklidischen Abstands in Merkmals-Vergleichen für herkömmliche Deskriptoren.
In [AS11] wurden die Ausführungszeiten für Detektion und Deskriptoren von verschiedenen Algorithmen untersucht. In Grafik 2.14 sind die Berechnungszeiten für 1000 Merkmale auf einem Intel Core i7-3770 aufgeführt, dabei sind ORB, BRISK und AKAZE Verfahren mit binärem Deskriptor.

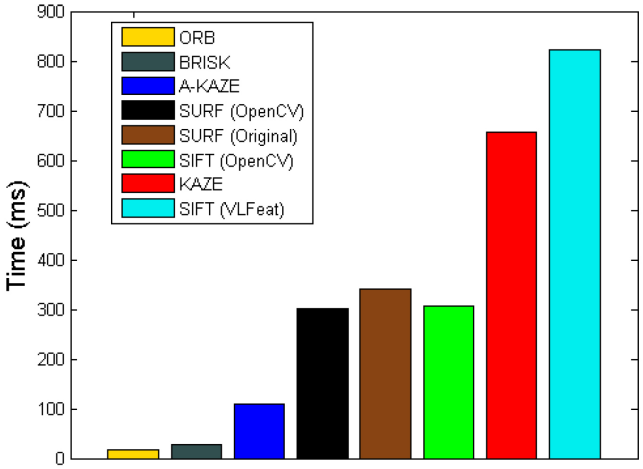


Abbildung 2.14: Berechnungszeiten für 1000 Merkmale [AS11]

Hier ist zu erkennen, dass diese Verfahren um mehrere Größenordnungen bessere Bildraten liefern als herkömmliche Algorithmen mit Fließkomma-Deskriptoren.

2.3 Feature Detektoren

2.3.1 FAST - Features from Accelerated Segment Test

Der Features from Accelerated Segment Test (FAST) Algorithmus [RD06] ist ein Feature Detektor der mittels Pixel-Vergleichen einen Merkmalspunkt finden und hinsichtlich seiner Kanteneigenschaften klassifizieren kann. Aufgrund des einfachen und effizienten Aufbaus ist dieser Algorithmus schneller als andere Feature-Detektoren und kann für Echtzeitanwendungen eingesetzt werden. Für die Berechnung werden in einem 7x7 Pixel Bildausschnitt 16 auf einem Kreis angeordnete Test-Punkte mit dem Mittelpunkt verglichen. Abbildung 2.15 zeigt das verwendete Pixel-Pattern. Für das mittlere Pixel p mit der Intensität I_p und einem Schwellwert t erfolgt für jeden Test-Punkt $x \in \{1...16\}$ über die Formel 2.2 eine Einteilung in die Klassen hell, dunkel oder gleich.

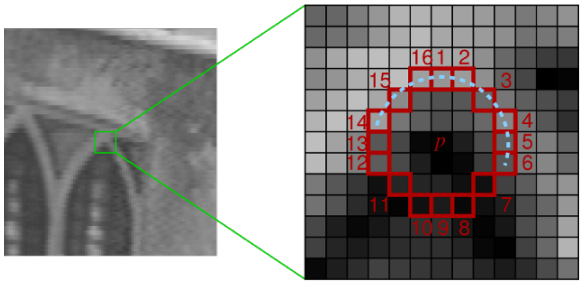


Abbildung 2.15: Fast Corner Detektor [RD06]

Falls auf dem Kreis n aufeinanderfolgende Pixel der gleichen Klasse hell oder dunkel zugeordnet werden können, wird der Pixel p als Merkmalspunkt klassifiziert. Der Fast Detektor mit $n = 9$ erzielt dabei nach Rosten und Drummond [RD06] die robustesten Ergebnisse und wird daher in den meisten Implementierungen verwendet. Um die Segment-Tests zu beschleunigen, haben die Autoren die Reihenfolge der Vergleiche mit maschinellem Lernen optimiert.

$$S_{p\to x} = \begin{cases} d, \ I_{p\to x} \leq I_p - t & (dark) \\ s, \ I_p - t < I_{p\to x} < I_p + t & (similar) \\ b, \ I_{p\to x} \geq I_p + t & (bright) \end{cases} \tag{2.2}$$

Um eine bessere Wiederholbarkeit und eine bessere Merkmals-Qualität zu erreichen, kann zusätzlich ein Non-Maxima Suppression Algorithmus angewendet werden. Dazu wird für jedes Merkmal ein Score über die Summe der Vergleichsantworten der hellen und dunklen Test-Punkte nach Formel 2.3 bestimmt.

$$score = max\Big(\sum_{x \in S_{bright}} |I_{p \to x} - I_p| - t, \sum_{x \in S_{dark}} |I_p - I_{p \to x}| - t\Big) \tag{2.3}$$

Über diesen Wert kann das Maximum in einem 3x3- oder 5x5-Pixel Fenster ausgewählt werden, um nur die Merkmale mit dem besten Score als Merkmal zurück zu liefern.

2.3.2 AGAST - Adaptive and Generic Accelerated Segment Test

Der Adaptive and Generic Accelerated Segment Test (AGAST) Detektor [MHB+10] ist eine Erweiterung des FAST Detektors mit adaptiven und generischen Segment-Vergleichen durch eine optimierte Struktur des Entscheidungsbaums.
Dazu wurden der Konfigurationsraum des originalen FAST Algorithmus um 2 weitere Zustände erweitert:

$$S_{p \to x} = \begin{cases} d, \ I_{p \to x} \leq I_p - t & (dark) \\ \overline{d}, \ I_{p \to x} > I_p - t \wedge S'_{p \to x} = u & (not\ dark) \\ s, \ I_{p \to x} > I_p - t \wedge S'_{p \to x} = \overline{b} & (similar) \\ s, \ I_{p \to x} < I_p + t \wedge S'_{p \to x} = \overline{d} & (similar) \\ \overline{b}, \ I_{p \to x} < I_p + t \wedge S'_{p \to x} = u & (not\ bright) \\ b, \ I_{p \to x} \geq I_p + t & (bright) \end{cases} \tag{2.4}$$

Dabei ist $S'_{p \to x}$ der vorhergehende Zustand, I ist die Helligkeit des Pixels und t der verwendete Schwellwert. Die Variable u beschreibt den Fall, dass der Zustand noch unbekannt ist.
Durch diese Erweiterung ist der Entscheidungsbaum vollständig (keine falsch positiven und falsch negativen Antworten) und lässt durch Anpassung des Vergleichsbaums deutlich kürzere Antwortzeiten zu.
Zusätzlich konnten im AGAST Detektor zwei Vergleichsbäume kombiniert werden, um die Kantendetektion automatisch an Bildregionen mit unterschiedlichen Strukturen anzupassen. Dadurch wird der Detektor generischer und muss nicht für einzelne Datensätze neu angepasst werden.
In Abbildung 2.16 ist die Kombination und Umschaltung mit mehreren Entscheidungsbäumen dargestellt. Der AGAST Detektor schaltet dabei auf spezifische Teile des Entscheidungsbaums, sobald die Pixel-Nachbarschaft sich ändert. Der linke Teil des Baumes ist auf homogene Bildbereiche optimiert und der rechte Teil wird in texturierten Regionen verwendet.
Der AGAST Detektor bietet mit dem 16-Pixel Vergleichspattern (OAST-9) im Vergleich zum FAST9 Pattern um 13% schnellere Antwortzeiten und eine robustere, generische Kantenerkennung.

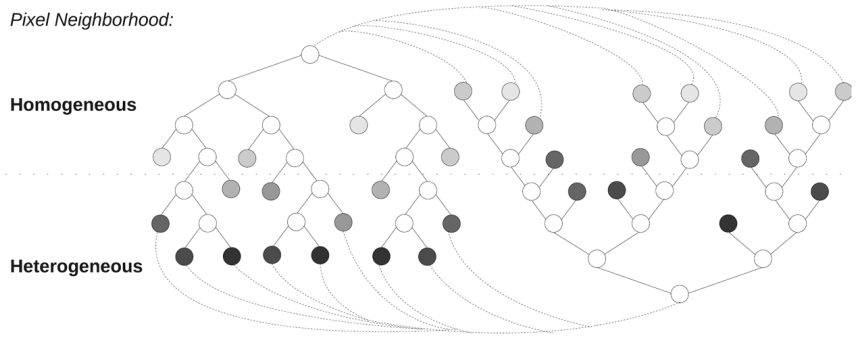


Abbildung 2.16: AGAST - Verwendung und Umschaltung mehrerer Entscheidungsbäume [MHB+10]

2.4 Feature Deskriptoren

2.4.1 SIFT - Scale-invariant Feature Transform

Der Scale-invariant Feature Transform Algorithmus [Low99] [Low04] ist eine Kombination aus Merkmalsdetektor und Merkmalsdeskriptor und wurde mit dem Ziel entwickelt, Bildvergleiche skalierungs- und rotationsinvariant auszuführen.

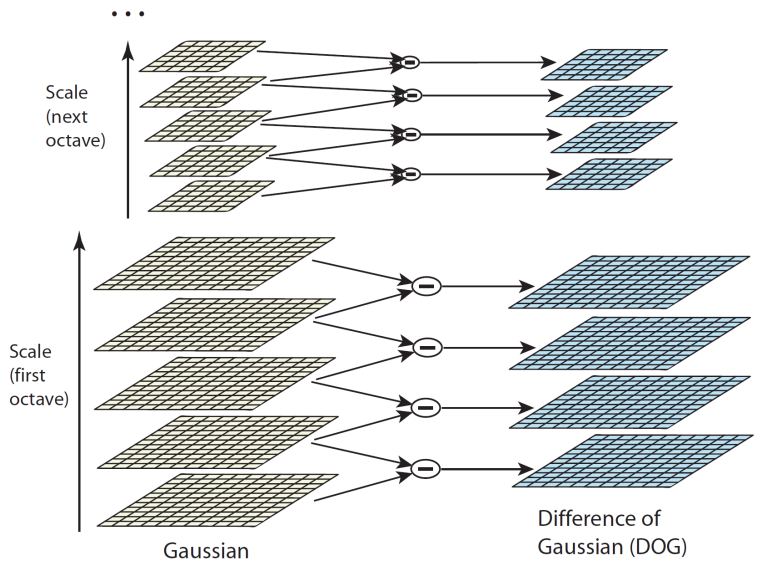


Abbildung 2.17: SIFT Skalenraum und Difference of Gaussian(DoG) Detektor [Low04]

Zur Merkmalsdetektion wird ein Skalenraum, wie in Abbildung 2.17 gezeigt, durch Skalierung und Gauss-Filterung aufgebaut. Jede Oktave besteht da-

bei aus mehreren Gauss-Faltungen. Nach jeder Oktave wird das Bild um den Faktor 2 skaliert. Daraus können die Merkmalspunkte durch Differenz der Gauss-Filterstufen (DoG) und durch Bestimmung der Extrema der 3x3 Nachbarschaft in der aktuellen, vorherigen und nächsten Skalen-Stufe bestimmt werden.

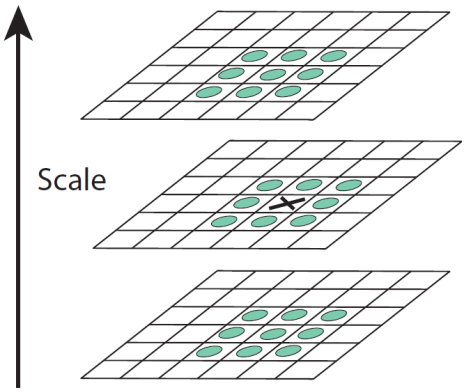


Abbildung 2.18: Detektion durch Bestimmung der Skalenraum Extrema [Low04]

In der neueren Version des SIFT Algorithmus [Low04] wird zur Bestimmung der Subpixel-genauen Merkmalsposition mit der Skalenraum-Funktion $D(x, y, \sigma)$ eine 3D-quadratische Funktion über Taylor-Reihen Minimierung angepasst:

$$D(x) = D + \frac{\partial D^T}{\partial x}x + \frac{1}{2}x^T\frac{\partial^2 D}{\partial x^2}x \tag{2.5}$$

Der genaue Ablauf der Koordinaten-Interpolation ist in Kapitel 4.2 näher erläutert. Im Anschluss werden die erkannten Merkmalspunkte mit einem Kontrast-Schwellwert $D(\hat{x}) > 0.03$ gefiltert, um nur die besten Ergebnisse zu erhalten.
Damit die Beschreibung des Merkmals invariant gegenüber Rotation ist, wird für jedes Merkmal eine Orientierung bestimmt.
In einem Fenster um jedes Merkmal wird zu jedem Abtastpunkt $L(x, y)$ eine Gradientengröße $m(x, y)$ und eine Orientierung $\sigma(x, y)$ nach

$$\begin{aligned} m(x,y) &= \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \\ \sigma(x,y) &= \tan^{-1}(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}) \end{aligned} \tag{2.6}$$

berechnet.
Nun wird mit den Orientierungen der Gradienten ein Histogramm mit 36 Bins erstellt. Dabei werden die Orientierungen über eine Gauss-Faltung um

den Mittelpunkt gewichtet, damit die Werte in den Ecken der Merkmalsregion nicht in die Berechnung mit eingehen. Abbildung 2.19 zeigt die Erstellung eines Orientierungshistogramms. Aus dem Maximum des Histogramms kann dann die dominante Richtung und damit die Merkmalsorientierung bestimmt werden.

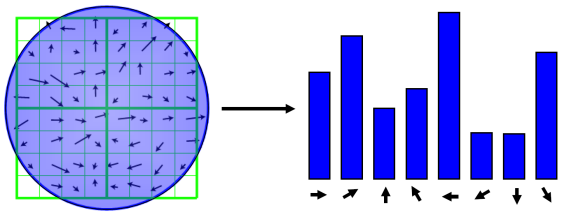


Abbildung 2.19: SIFT Orientation Histogramm [Low04]

Der SIFT Deskriptor zur Beschreibung eines Merkmals besteht aus 32-128 Gleitkommazahlen und wird ebenfalls aus Gauss-gewichteten Gradientenhistogrammen in der Merkmalsumgebung bestimmt. Dafür wird das Deskriptorfenster in 2x2 bzw 4x4 große Unterregionen von 8 Bin Orientierungshistogrammen geteilt.
Damit der Deskriptor unabhängig von der Merkmalsorientierung ist, werden die berechneten Gradientenwerte um die zuvor berechnete Merkmalsorentierung rotiert.

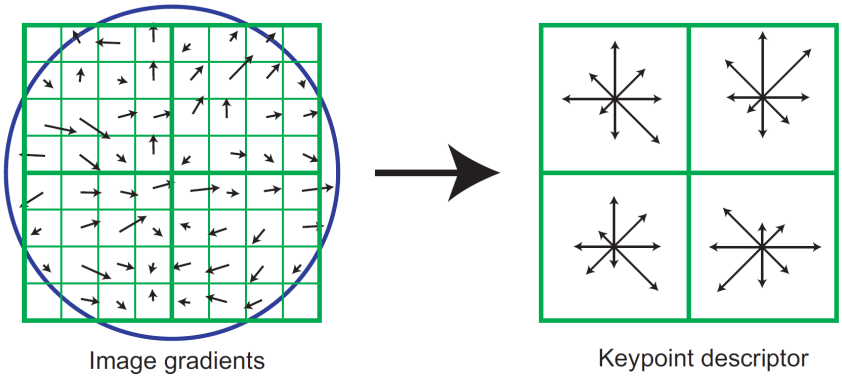


Abbildung 2.20: SIFT Deskriptor mit 2x2x8 Abtastpunkten [Low04]

Abbildung 2.20 zeigt den SIFT Deskriptoraufbau mit 2x2 Unterregionen. Der in Lowes Paper [Low04] vorgestellte Algorithmus nutzt ein 4x4x8 Pattern mit 128 Deskriptorwerten.

2.4.2 SURF - Speeded Up Robust Features

Speeded Up Robust Features (SURF) [BTVG06] ist eine Weiterentwicklung des SIFT Algorithmus zur Vereinfachung und Beschleunigung der Merkmalsextraktion. Das Verfahren basiert auf den gleichen Ideen wie SIFT, kann aber durch Einsatz von Integral-Bildern zur Berechnung der Gauss-Faltungen und durch einen kompakteren Deskriptor die Berechnungszeit verkürzen. Zur Merkmals-Detektion wird ein Fast-Hessian Detektor vorgestellt - eine Annäherung des Hessian-Laplace Detektors über Box-Filter. Die Box-Filter können die Gauss'sche partielle Ableitungen zweiter Ordnung annähern und sind durch Integral-Bilder unabhängig von der Filter-Größe sehr effizient und schnell zu berechnen. Zur Abbildung des Skalenraums können damit anstelle einer Bild-Skalierung die Box-Filter Größen variiert werden. Die Grafik 2.21 zeigt die Annäherung Gauss'sche partielle Ableitungen durch Box-Filter.

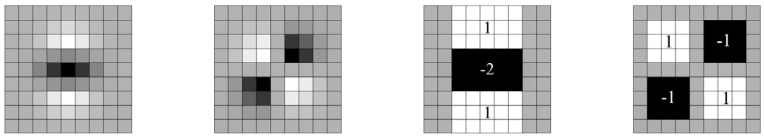


Abbildung 2.21: Links: Gauß'sche partielle Ableitungen zweiter Ordnung in Y- und XY-Richtung, Rechts: Annäherungen mit Box-Filtern [BTVG06]

Die Orientierung eines SURF Merkmals wird über die dominante Orientierung aus Haar-Wavelet Antworten in X- und Y- Richtung berechnet. Zusätzlich ist mit dem U-SURF auch eine schneller zu berechnende Variante ohne Orientierung verfügbar.
Der SURF Deskriptor teilt die Merkmalsumgebung in quadratische 4x4 Unterregionen. In den Unterregionen werden aus 5x5 Test-Punkten die Haar-Wavelet Antworten d_x und d_y in X- und Y-Richtung bestimmt. Die Wavelet Antworten werden für jede der 4x4 Unterregionen aufsummiert und zusätzlich auch die Absolutwerte $|d_x|$ und $|d_y|$ berechnet.

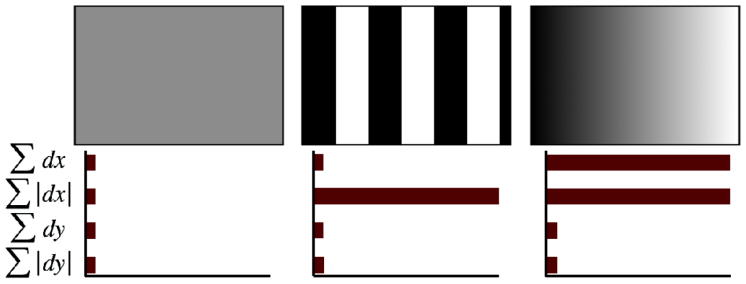


Abbildung 2.22: Deskriptor-Einträge aus Haar-Wavelet Antworten und die Antworten für verschiedene Bildstrukturen [BTVG06]

Dadurch erhält jede Unter-Region einen 4-dimensionalen Vektor v der Form

$$v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|). \tag{2.7}$$

Aus den Vektoren der 4x4 Regionen wird dann ein Deskriptor mit 64 Einträgen erstellt. Abbildung 2.22 zeigt die Bestandteile des Deskriptors und die Antworten für unterschiedliche Bildstrukturen.
In [BTVG06] wurden der SURF Detektor und Deskriptor einzeln evaluiert und mit SIFT und anderen Verfahren verglichen. Durch die Optimierungen ist der Fast-Hessian Detektor 3 mal schneller zu berechen als der SIFT DoG Detektor. Dabei bietet der SURF Detektor eine gleiche oder bessere Wiederholbarkeit.

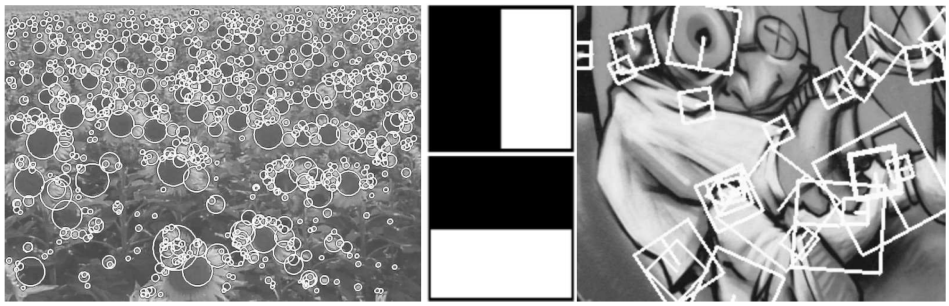


Abbildung 2.23: Links: erkannte Merkmale in einem Sonnenblumenfeld, Mitte: Haar-Wavelet Typen, rechts: Deskriptorfenster-Größen für verschiedene Skalierungen [BTVG06]

Der SURF Deskriptor erzielt eine sehr gute Präzision und Wiedererkennung im Vergleich zu den anderen getesteten Verfahren mit bis zu 10% besseren Ergebnissen im Vergleich zu SIFT. Dabei benötigt der SURF Algorithmus im Test 2,9x weniger Zeit zur Erstellung des Deskriptors als SIFT.

2.4.3 ORB - Oriented FAST and Rotated BRIEF

Die Oriented FAST and Rotated BRIEF (ORB) Merkmalserkennung wurde von Rublee [RRKB11] vorgestellt und ist eine Entwicklung von Willow Garage. Der Algorithmus ist eine Kombination und Weiterentwicklung des effizienten FAST Detektors und des binären Merkmals-Deskriptors BRIEF. ORB nutzt einen 31x31 Pixel Ausschnitt der Umgebung des zu beschreibenden Bildmerkmals, um darin 256 Pixel-Vergleiche mit ausgewählten Vergleichspaaren auszuführen. Die Ergebnisse der Vergleiche dienen als Deskriptor.

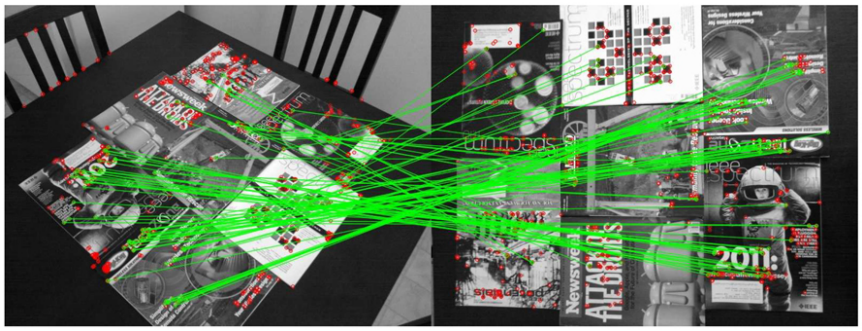


Abbildung 2.24: ORB Matching Ergebnis mit Blickwinkelveränderung [RRKB11]

Zur Auswahl der Vergleichspaare stellen die Autoren eine Trainings-Methode zum Erlernen der besten Binärvergleiche vor. Dazu werden 300K Test-Punkte eines Bild-Datensatzes verwendet, um die optimale Verteilung der Vergleichspaare zu ermitteln. Abbildung 2.25 zeigt das erlernte ORB Deskriptor-Pattern mit guter Diversität und reduzierter Korrelation zwischen den Vergleichspunkten.
Um eine Rotations-Invarianz der Merkmals-Beschreibung zu erreichen, stellt ORB eine Erweiterung des FAST Detektors zur Bestimmung der Merkmals-Orientierung über ein Intensity Centroid vor. Dabei werden zunächst auf dem verwendeten Bildausschnitt die Momente m in X-Richtung und Y-Richtung berechnet:

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \tag{2.8}$$

Mit den Momenten kann dann der Centroid des Ausschnitts bestimmt werden:

$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}) \tag{2.9}$$

Die Orientierung des Ausschnitts kann damit über

$$\theta = atan2(m_{01}, m_{10}) \tag{2.10}$$

berechnet werden.

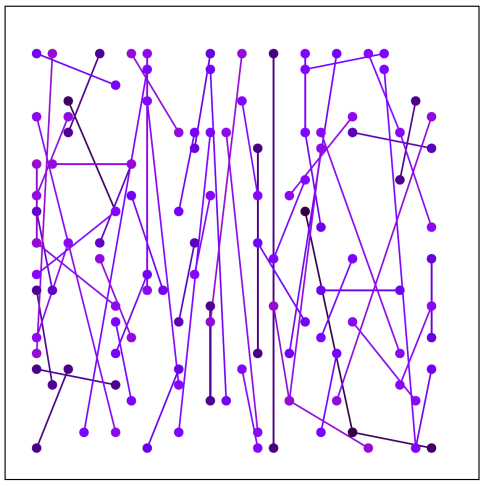


Abbildung 2.25: ORB Deskriptor Pattern [RRKB11]

Bei der Deskription des Bildausschnitts wird die Orientierung des Merkmals verwendet, um das angepasste BRIEF Deskriptor-Pattern entsprechend zu rotieren. In Abbildung 2.26 ist die Rotations-Invarianz des ORB-Verfahrens gezeigt.

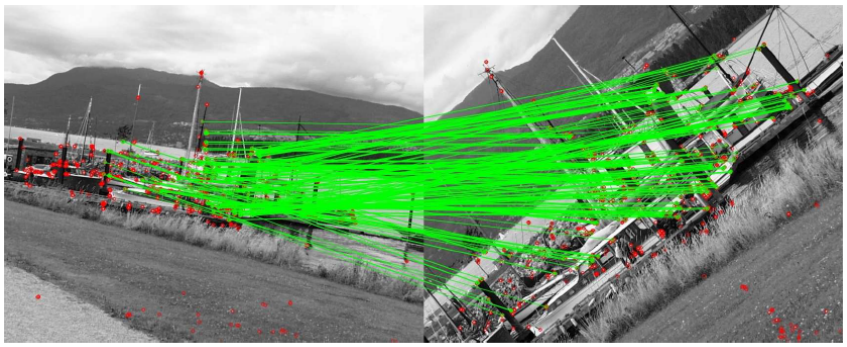


Abbildung 2.26: ORB Rotations-Invarianz durch Deskriptor-Pattern Rotation [RRKB11]

ORB verwendet Skalen-Pyramiden [KM08], um Merkmals-Punkte skaleninvariant zu detektieren und zu beschreiben. Als Vorverarbeitungsschritt wird dazu zunächst eine Skalen-Pyramide für die gewünschte Anzahl an Ebenen durch Skalieren des Eingangsbildes mit einem definierten Faktor aufgebaut. Der FAST Detektor wird dann auf allen Ebenen angewendet, um Merkmale

in verschiedenen Skalen zu finden und somit einer Oktave zuzuordnen. Der ORB Deskriptor verwendet die Oktaven-Information des Merkmals, um das Merkmal in der entsprechenden Pyramiden-Ebene und damit unabhängig von der Skalierung zu beschreiben.
In [RRKB11] wurde ORB mit den SIFT und SURF Verfahren verglichen. Die Evaluation der Ausführungszeiten hat ergeben, dass ORB mit 15,3 ms für die Berechnung von 1000 Merkmalen über eine Größenordnung schneller ist als SURF (217,3 ms) und zwei Größenordnungen schneller ist als SIFT (5228,7 ms). Dabei kann ORB in den meisten Datensätzen bessere oder gleichwertige Ergebnisse der Detektor- und Deskriptor-Qualität erzielen.
Der ORB-Algorithmus ist ein Merkmalsextraktions-Verfahren mit einer sehr geringen Berechnungszeit trotz einer guten Erkennungsrate und Invarianz zu Blickwinkeländerungen und Bildrotationen.

2.4.4 FREAK - Fast Retina Keypoint

Der Fast Retina Keypoint Deskriptor [AOV12] ist ein binärer Merkmals-Deskriptor, der von dem menschlichen visuellen System inspiriert ist. Die Autoren haben dazu die Topologie und den Aufbau der menschlichen Retina betrachtet. Die Größe des rezeptiven Feldes der Retina erhöht sich mit dem radialen Abstand von der Foveola. Dabei wird die räumliche Verteilung der Ganglienzellen mit dem Abstand zur Foveola exponentiell kleiner. Abildung 2.28 zeigt den Aufbau der menschlichen Retina.

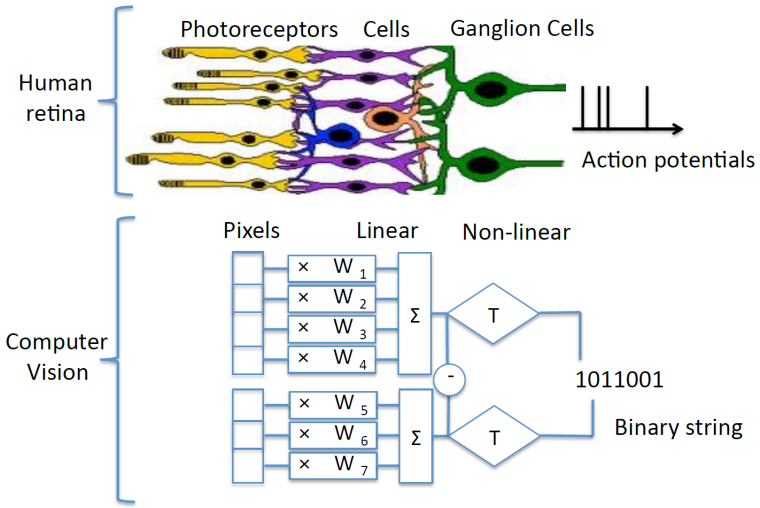


Abbildung 2.27: Von der menschlichen Retina zur Bildverarbeitung [AOV12]

Inspiriert von diesem Aufbau haben die Autoren einen Merkmals-Deskriptor-Pattern entwickelt. Das FREAK-Pattern ist in Abbildung 2.29 dargestellt. Die Dichte der Pattern-Punkte wird zur Mitte des Patterns hin höher, somit

ergibt sich Retina-ähnliche Verteilung. Zusätzlich werden die Filtergrößen für die Gauss-Faltung zur Glättung der Bildregionen variiert. Die Autoren haben hier herausgefunden, dass die angepassten Größen der Faltungskerne und überlappende rezeptive Felder die Performanz des Deskriptors erhöhen.

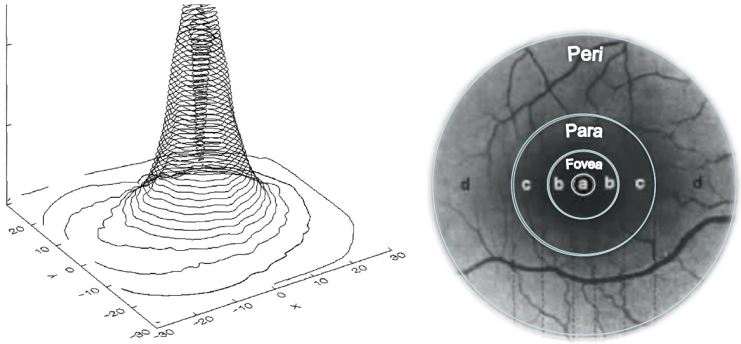


Abbildung 2.28: Dichte der Ganglienzellen und Aufbau der menschlichen Retina [AOV12]

Der FREAK Deskriptor nutzt 512 Vergleichspaare und hat somit eine Länge von 512 Bit. Der Deskriptor wird mit einem Coarse-To-Fine Aufbau erstellt, bei dem die Reihenfolge der Deskriptor Bytes aufgrund des Retina-Aufbaus mit dem Detailgrad übereinstimmt. Dadurch lassen sich beim Deskriptor-Vergleich schneller inkorrekte Matches ausschließen.

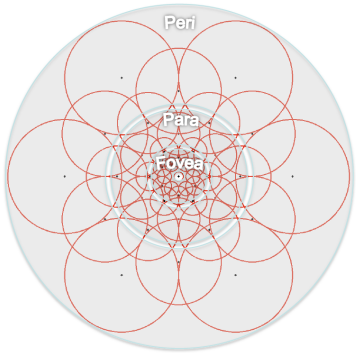


Abbildung 2.29: FREAK Deskriptor Pattern [AOV12]

Zur Bestimmung der Deskriptor Orientierung nutzt FREAK eine symmetrische Verteilung von 45 Vergleichspaaren, an deren Position die Summe der lokalen Gradienten gebildet werden.
In dem Paper [AOV12] erzielt der FREAK Deskriptor sehr gute Ergebnisse in den Deskriptor-Evaluations-Tests. Der Deskriptor ist robust gegenüber allen Bild-Deformationen und bietet im Vergleich zu SIFT, SURF und BRISK eine deutlich kürzere Berechnungs- und Matching-Zeit.

Kapitel 3

Verwandte Arbeiten

In diesem Kapitel sollen andere Arbeiten und bestehende Ansätze zur Beschleunigung von SLAM Systemen vorgestellt werden. Dazu sollen auch Implementierungen von Merkmalserkennungen auf FPGA-Architekturen betrachtet werden.

3.1 FPGA-based ORB Feature Extraction for Real-Time Visual SLAM

In [FZYL17] wurde von W.Fang et al. eine FPGA-basierte ORB-Merkmalserkennung für visuellen SLAM vorgestellt. Die Autoren nutzen ein Visual Inertial SLAM System das auf MSCKF und ORB-SLAM basiert. Das Profiling auf einer ARM v8 Architektur hat ergeben, dass die Merkmalsextraktion mit über 50% Auslastung der CPU der rechenintensivste Teil des vorgestellten Systems ist. Daher wird in der Arbeit eine FPGA Implementierung der ORB Merkmalsextraktion auf einem Altera Stratix V FPGA vorgestellt, um die SLAM Anwendung zu beschleunigen.

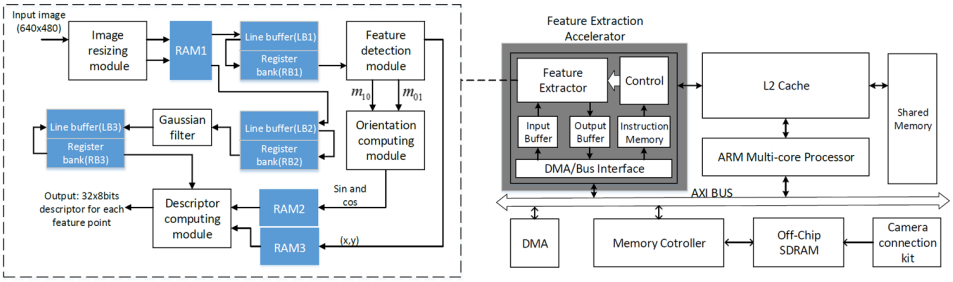


Abbildung 3.1: Architektur eines Echtzeit-SLAM Systems mit ORB-Merkmalserkennung in Hardware [FZYL17]

Die Implementierung unterstützt Bildauflösungen von 640x480 Pixel und

umfasst die Skalenraum-Erstellung für 2 Skalen. Der FAST Detektor mit Orientierungsberechnung und das Deskriptor-Modul sind mit einem 2-stufigen System aus Linienspeichern umgesetzt. Die Merkmals-Orientierung θ wird über die Momente m_{01} und m_{10} des kreisförmigen Ausschnitts nach

$$\theta = \arctan \frac{m_{01}}{m_{10}} \tag{3.1}$$

$$\sin\theta = \frac{m_{01}}{\sqrt{m_{10}^2 + m_{01}^2}} \quad \cos\theta = \frac{m_{10}}{\sqrt{m_{10}^2 + m_{01}^2}} \tag{3.2}$$

bestimmt. Wang hat dabei den Einfluss der Wortlänge zur Speicherung der Momente und den daraus resultierenden Fehler analysiert, um die Hardware Implementierung zu optimieren. Für die Implementierung wurde die Wortlänge von 21Bit auf 8Bit reduziert.

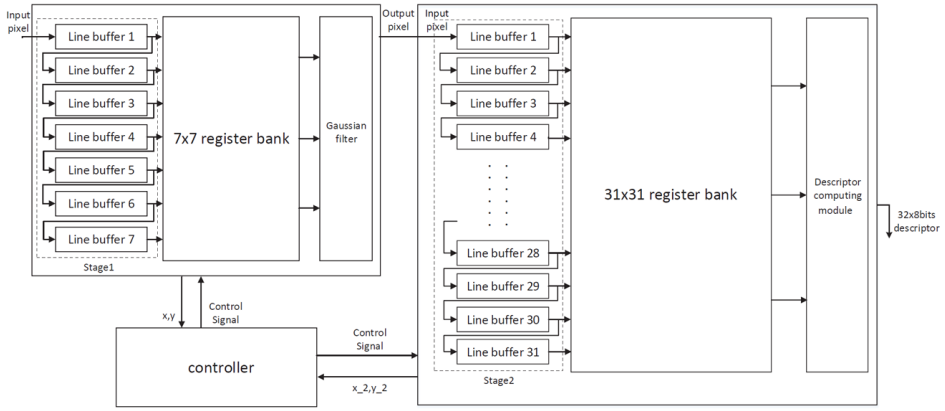


Abbildung 3.2: Hardware Architektur der 2-stufigen ORB-Merkmalserkennung [FZYL17]

Abbildung 3.2 zeigt die Hardware Architektur der ORB-Merkmalserkennung. Das System nutzt eine 7x7 Pixel Registerbank mit Linienspeicher zur Gauss-Filterung der Eingangsbilder und eine 31x31 Pixel Registerbank mit Linienspeicher zur Deskriptorerstellung.
Die in [FZYL17] vorgestellte ORB-Merkmalserkennung erreicht auf dem Altera Stratix V FPGA eine Taktrate von 203MHz und eine Latenz von 14,8ms zur Bearbeitung von Bildern in VGA-Auflösung. Das entspricht einer Bildrate von 67 Bildern pro Sekunde und ist damit 51% schneller als das ARM v8 Vergleichssystem und 41% schneller als ein Intel Core i5.
Abbildung 3.3 zeigt die Hardware Performance und den Vergleich zu den Referenz-Architekturen.
Die vorgestellte Arbeit zeigt die gute Parallelisierbarkeit der ORB-Merkmalsextraktion auf einer FPGA-Architektur und den Einsatz in einem heterogenen System mit ARM Prozessor. Allerdings wurden zur Umsetzung in

	Clock Freq. (GHz)	Latency (ms)	Throughput (FPS)	Engergy (mJ/frame)
Proposed design	0.203	14.8	67	68
ARM Krait	2.26	30	33	75
Intel Core i5	2.9	25	40	400
Improvement vs ARM	–	51%	103%	9%
Improvement vs Intel	–	41%	68%	83%

Abbildung 3.3: Hardware Performance Vergleich mit ARM Krait und Intel Core i5 [FZYL17]

Hardware einige Vereinfachungen vorgenommen, die sich eventuell negativ auf den Einsatz in einem SLAM-System auswirken. So wurden lediglich 2 Skalenstufen bei einer vergleichsweise geringen Bildauflösung implementiert. Der OpenCV ORB wird zum Einsatz in SLAM Systemen meist mit 4-8 Skalenstufen konfiguriert, um eine gute Skalen-Invarianz zu erreichen.
Der Einfluss der Vereinfachungen auf die Lokalisierung und Ergebnisse beim Einsatz des genannten Verfahrens in SLAM Systemen wurden leider nicht gezeigt. Die Arbeit gibt ebenfalls keine Aussagen zur Qualität der Merkmale wie z.B. die Genauigkeit oder Wiederholbarkeit des implementierten Detektors / Deskriptors.

3.2 FPGA Acceleration of Multilevel ORB Feature Extraction for Computer Vision

In der Arbeit [WKBD17] wurde eine FPGA Implementierung und Beschleunigung einer Multi-Skalen ORB Merkmals-Extraktion für Bildverarbeitung und visuelles SLAM vorgestellt. In einer vorausgegangenen Arbeit [WKD15] implementieren die Autoren auf einer FPGA-Architektur eine ORB Merkmals-Erkennung mit Harris-Stephens Detektor, der im originalen ORB Verfahren zur Berechnung der Merkmals-Antworten verwendet wird. In [WKBD17] wird dieses Verfahren mit einer Multi-Skalen Detektion und Deskription erweitert, um eine Skalen-Invarianz zu erhalten und die Ergebnisse des Verfahrens für SLAM Anwendungen zu verbessern. Dazu wurden zunächst die benötigten Skalentiefen bzw. die Anzahl der benötigten Pyramidenlevel für verschiedenen Datensätze untersucht. Eine Oktave entspricht dabei einer Bild-Skalierung, die wiederum eine variable Anzahl an Pyramidenleveln durch Unterabtastung enthalten kann.
Die Abbildungen 3.4 und 3.5 zeigen die Anzahl der Matches mit steigender Anzahl von Pyramidenleveln und Pyramidenleveln pro Oktave auf dem Kitti [GLU12] Datensatz und dem Oxford [MTS+05] Datensatz.

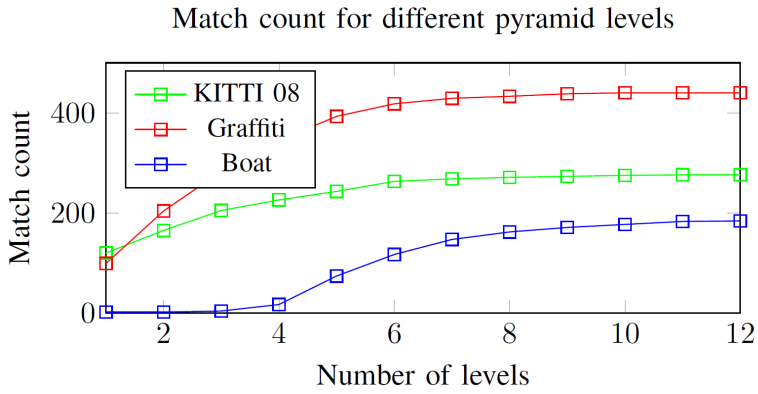


Abbildung 3.4: Anzahl der Matches mit steigender Anzahl von Pyramidenleveln [WKBD17]

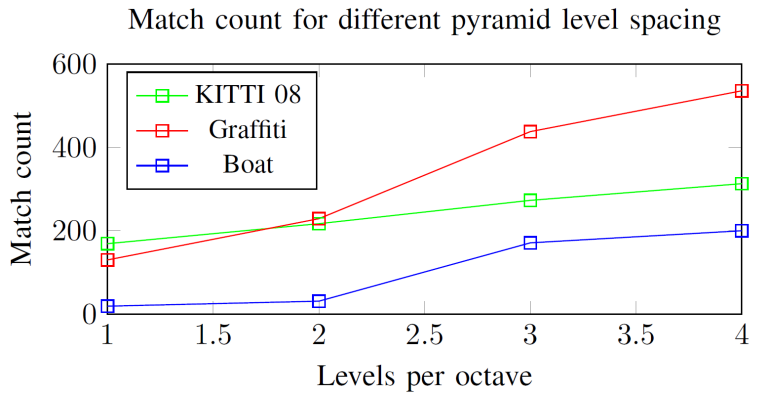
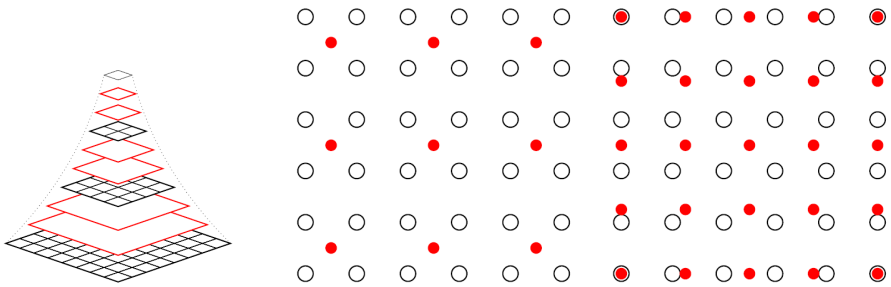


Abbildung 3.5: Anzahl der Matches mit steigender Anzahl von Pyramidenleveln pro Oktave [WKBD17]

In der Arbeit wurde zur Implementierung ein System mit 3 Oktaven und 3 Leveln pro Oktave durch Unterabtastung mittels bilinearer Interpolation mit 1/2 und 4/5 Downsampling-Filtern ausgewählt. Der verwendete Aufbau der Skalenpyramide und die Downsampling-Filter sind in Grafik 3.6 dargestellt.

Die vorgestellte Merkmalsextraktion, von den Autoren Tarsier genannt, verwendet mehrere parallele ORB-Instanzen zur Detektion und Deskription über 9 Skalen. Jedes ORB-Modul nutzt dabei einen 37x37 Pixel Schieberegister Speicher und benötigt 266 Takte zur Berechnung des Deskriptors eines Merkmals.

Die Merkmals-Orientierung wird in dem gezeigten System über Look-up Tabellen für die Acrtan und Sinus/Cosinus Berechnung realisiert. Die Look-up Tabellen haben dabei eine Größe von 64 Einträgen. Eine genaue Aussage über die Qualität / Genauigkeit der Winkel-Berechnungen wird nicht ge-



(a) (b) (c)

Abbildung 3.6: (a) Image Pyramide (Oktaven in schwarz, Level durch Unterabtastung in rot) (b) Unterabtastung mit 1/2 Filter (c) Unterabtastung mit 4/5 Filter [WKBD17]

troffen.
In der Arbeit wird das Tarsier System auf einem Altera Arria V FPGA implementiert und nutzt eine PCIe Schnittstelle zur Kommunikation mit dem Host-Computer. Abbildung 3.7 zeigt das System mit mehreren parallelen Einheiten für die 9 Skalen-Level.

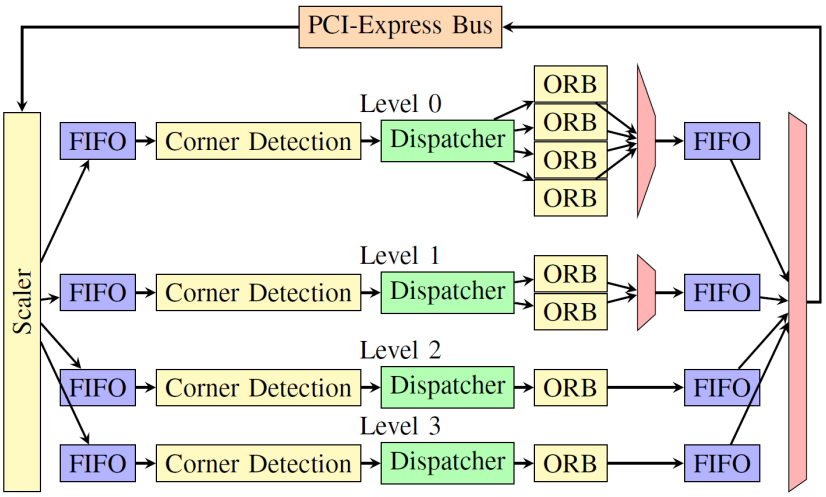


Abbildung 3.7: Tarsier System Übersicht [WKBD17]

Zur Evaluation der Performanz wurde die FPGA Implementierung mit einem Intel Core i5 2500k System mit 970GTX GPU verglichen. Abbildung 3.8 zeigt den Ressourcen-Verbrauch auf dem Altera Arria V FPGA, sowie die Ausführungszeiten und Bildraten der evaluierten Systeme bei Full-HD Auflösung. Zusätzlich wurde der Energieverbrauch abgeschätzt.

Logic Elements	206,000
Registers	231,973
Arria V ALMs	114,049 (83%)
DSP Elements	449 (43%)
Memory Bits	8,579,952 (49%)
f_{max} (slow model)	150 MHz
f_{max} (fast model)	230 MHz
TDP	5340 mW

(a)

Device	Throughput	Frame Latency	1080p FPS	Energy per frame (approx)
CPU (1 thread)	28 MPix/s	74.1 ms	13.5	405 mJ
CPU (4 threads)	100 MPix/s	74.1 ms	48.2	430 mJ
GPU	40 MPix/s	52.1 ms	19.2	15.5 J
Tarsier (slow model)	150 MPix/s	13.8 ms	72.3	74 mJ
Tarsier (fast model)	230 MPix/s	9.1 ms	110.9	57 mJ

(b)

Abbildung 3.8: (a) Benötigte Hardware Ressourcen (b) Performanz des Tarsier-Systems im Vergleich zu CPU/GPU [WKBD17]

Die Ergebnisse zeigen, dass die vorgestellte ORB Multi-Skalen Merkmalsextraktion auf dem FPGA sehr gute Bildraten von über 60 Bildern/s erreicht und damit derzeit eine der schnellsten ORB-Implementierungen mit vollständiger Multi-Skalen Unterstützung ist. Die Arbeit verwendet einige Vereinfachungen und Optimierungen wie z.B. Look-up Tabellen und eine Unterabtastung zur Skalenpyramiden-Konstruktion. Der Einfluss dieser Optimierungen auf die Genauigkeit wurde leider nicht gezeigt. So wäre z.B. ein Vergleich der Detektor/Deskriptor Ergebnisse mit der ursprünglichen OpenCV ORB Implementierung oder anderen Verfahren interessant gewesen, um die Ergebnisse im Vergleich zu dem Stand der Technik einzuordnen.
Die Arbeit zeigt aber die guten Beschleunigungsmöglichkeiten der ORB-Merkmalsextraktion auf einer FPGA-Architektur. Die Analyse des entwickelten Systems mit Datensätzen zur Evaluation von Detektoren und Deskriptoren sowie eine Evaluation in den genannten Einsatzgebieten wie z.B. der ORB-SLAM Navigation ist in der Arbeit nicht ausgeführt worden.

3.3 Architecture Exploration of Intelligent Robot System using ROS-compliant FPGA Component

In [OYM+16] wurde der Einsatz von FPGA-Systemen zur Beschleunigung von rechenintensiven SLAM Algorithmen in einem verteilten mobilen Robotersystem untersucht. Es wurde eine Design Exploration einer 3D SLAM Architektur in ROS vorgenommen, um den Einfluss einer zusätzlichen externen Beschleuniger-Komponente für die Bildmerkmalsbestimmung zu evaluieren. Abbildung 3.9 zeigt ein ROS kompatibles Modell zur Integration eines FPGA-Systems in die ROS-Infrastruktur.

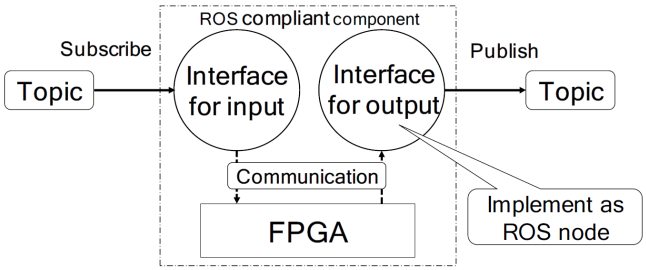


Abbildung 3.9: ROS compliant FPGA Komponente [OYM+16]

In der Arbeit wurde das RTAB-Map Visual SLAM als Fallstudie für die Design Space Exploration ausgewählt. Für die Untersuchung wurden die einzelnen Bestandteile (Nodes) des RTAB-Map Systems hinsichtlich ihrer Berechnungsdauer und ihres Kommunikationsbedarfs analysiert. Die Arbeit partitioniert dabei die Berechnungen in Bildaufnahme, Selbstlokalisierung, Kartenerstellung und Visualisierung. Abbildung 3.10 zeigt die Partitionierungsmöglichkeiten des RTAB-Map SLAM Systems.
Als Merkmalsextraktionsverfahren wurde das in RTAB-Map integrierte ORB-Verfahren genutzt, da es im Vergleich zu den anderen getesteten Verfahren SIFT und SURF die kürzesten Berechnungszeiten bietet. Das Ergebnis der Design Space Exploration für das verteilte Navigationssystem ist in Abbildung 3.11 dargestellt. Die Autoren zeigen, dass durch den Einsatz eines FPGA-Beschleunigers die Ausführungszeit auf der Roboter-Seite und auf der Cloud-Seite reduziert werden kann. Durch die Integration der Merkmalsextraktion kann in dem verteilten System ebenfalls der Kommunikationsaufwand von 46,3MB/s auf 480KB/s gesenkt werden, da nur noch die Merkmalspunkte und Deskriptoren übertragen werden.
Die Arbeit zeigt, dass die Integration eines FPGA-Beschleunigers möglich ist und dass durch den Einsatz einer Hardware-basierten Merkmalserkennung die Berechnungszeit der Selbstlokalisierung und der Kartenerstellung reduziert werden kann. Durch die in der Arbeit gewählte verteilte Architektur

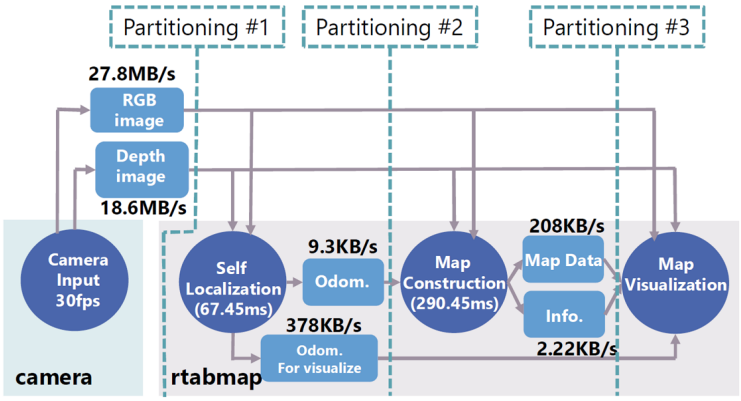


Abbildung 3.10: Partitionierungs-Kandidaten im RTAB-Map SLAM [OYM+16]

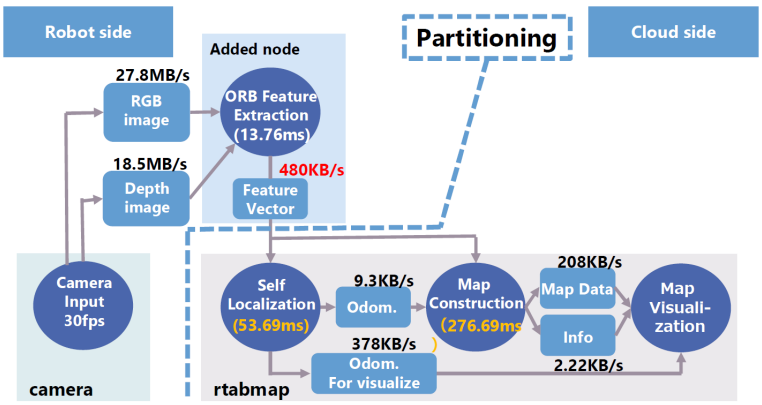


Abbildung 3.11: Ergebnis der Model-Level Design Space Exploration [OYM+16]

sind die Ergebnisse nur teilweise auf das in dieser Dissertation verwendete SLAM-System übertragbar, da hier der Kommunikations-Aufwand durch die direkte Integration des Beschleunigers in das SLAM-System geringer ist. In der Arbeit wurde der Einsatz eines FPGA-Beschleunigers in dem RTAB-Map System untersucht, allerdings wurde die Umsetzung und Implementierung eines solchen Systems nur als künftige Arbeit genannt und bleibt daher noch unerfüllt.

3.4 Navion: A Fully Integrated Energy-Efficient Visual-Inertial Odometry Accelerator

In dem Paper [SZC+18] wird mit Navion ein Energie-effizienter Beschleuniger für visuelle Odometrie für mobile autonome System wie z.B. Drohnen vorgestellt. Der Beschleuniger wird als ASIC-Design in einem integrierten Schaltkreis realisiert.
Der vorgestellte Navion Chip nutzt Daten einer Inertial Measurement Unit (IMU) und Mono- oder Stereobilder um die Flugbahn der Drohne und eine 3D-Karte der Umgebung zu berechnen. Dabei wird ein Visual Intertial Odometry (VIO) Algorithmus genutzt, der Bildmerkmale eines Merkmalsdetektors einsetzt, um diese mit einem Verfahren zur Bestimmung des optischen Flusses nach Lucas-Kanade [LK+81] zu verfolgen. Abbildung 3.12 zeigt die Architektur des Chips.

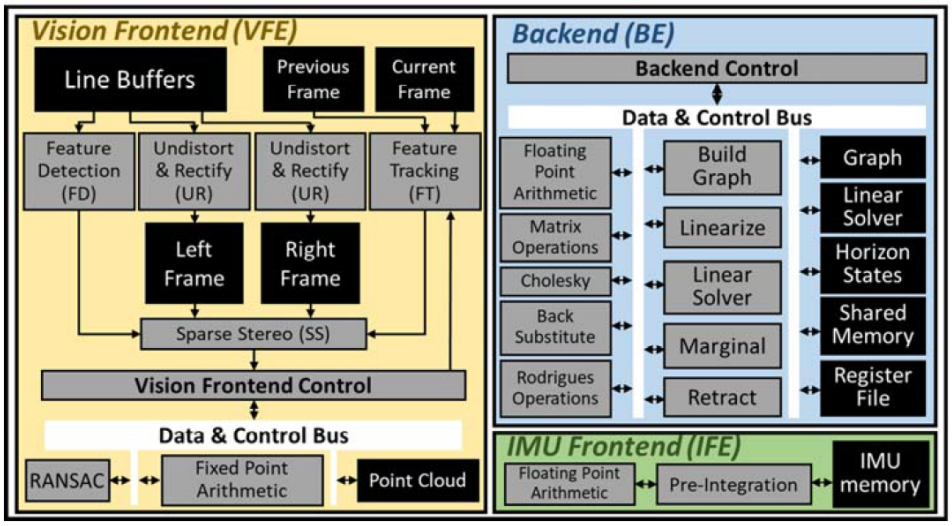
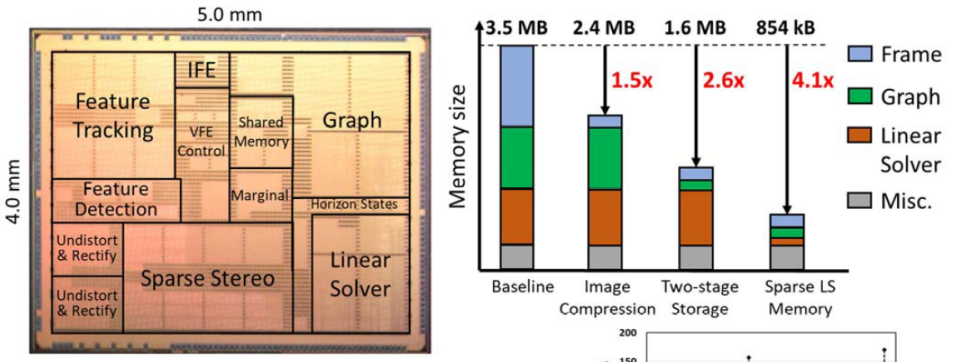


Abbildung 3.12: Navion Chip Architektur [SZC+18]

Das Design enthält Hardware Module zur Harris Corner [HS88] Merkmalsdetektion, die Lucas-Kanade [LK+81] Merkmalsverfolgung und ein Block-Matching Verfahren zur lokalen Stereo Tiefenberechnung. Die Flugbahn wird zur Laufzeit aus Keyframes über Graphen-Optimierung mit der Gauss-Newton Methode und einem Linear Solver berechnet und optimiert.
Der Chip wurde in einem 65nm CMOS Prozess gefertigt und kann Stereo-Bilder mit einer Auflösung 752x480 Pixeln bei einer Bildrate von 28 - 171 Bildern/s verarbeiten. Die 3D Karte kann dabei mit 16-90 Bilder/s aktualisiert werden. Abbildung 3.13 zeigt ein Foto des Halbleiter-Chips und die Spezifikation des Designs mit Speichergrößen, Bildraten sowie dem Energieverbrauch.



Technology	65nm CMOS	**Supply**	1 V
Chip area (mm²)	4.0 x 5.0	**Resolution**	752 x 480
Core area (mm²)	3.54 x 4.54	**Camera rate**	28 - 171 fps
Logic gates	2,043 kgates	**Keyframe rate**	16 - 90 fps
SRAM	854 kB	**Average Power**	24 mW
VFE Frequency	62.5 MHz	**GOPS**	10.5 – 59.1
BE Frequency	83.3 MHz	**GFLOPS**	1 – 5.7

Abbildung 3.13: Foto des Halbleiter-Chips und Chip Spezifikation [SZC+18]

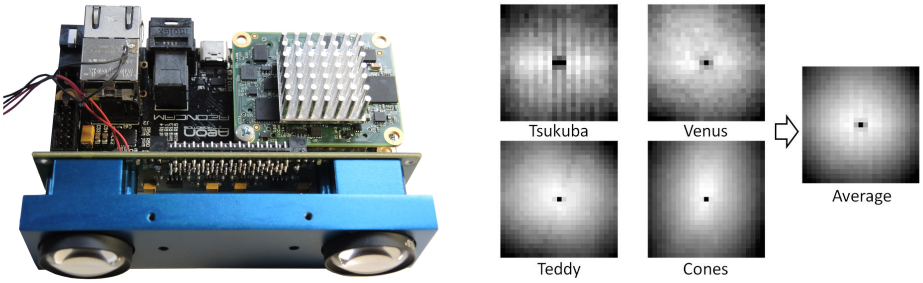
Die Ergebnisse zeigen, dass mit einer ASIC-Implementierung eine sehr gute Performanz bei geringer Stromaufnahme erreicht werden kann. Der vorgestellte Algorithmus ist dabei ideal auf die im Paper genannte Anwendung der visuellen Bewegungsverfolgung von leichten mobilen Drohnen-Systemen ausgelegt. Für diese Anwendung ist die Kombination aus einem einfachen Merkmalsdetektor und einem Verfahren zur Merkmalsverfolgung zielführend, da hier sehr hohe Bildraten erzielt werden können.
Allerdings ist es ohne Berechnung eines Merkmals-Deskriptor nicht möglich, Vergleiche von Bildmerkmalen auszuführen. Somit können mit diesem Ansatz kein Loop Closures erkannt werden oder die Wiederaufnahme der Bewegungsverfolgung nach einem Abbruch der visuellen Odometrie realisiert werden. In dieser Dissertation wird daher ein Verfahren der merkmalsbasierten SLAM-Lokalisierung mit Merkmalsvergleichen anhand von Deskriptoren [LM11] betrachtet, um auch SLAM-Anwendungen mit höheren Anforderungen abdecken zu können.

3.5 Vorarbeiten

3.5.1 Real-time Smart Stereo Camera based on FPGA-SoC

Als Vorarbeit zu dieser Dissertation wurde in [MMNB17] ein Stereo-Kamerasystem mit integrierte Stereo-Bildverarbeitung in einem FPGA-SoC entwickelt. Das Kamerasystem enthält einen Zynq XC7Z030 FPGA-SoC und zwei

Aptina AR0134 Bildsensoren mit HD-Auflösung. Der Verbindung zu einem Host-PC wird über USB 3.0 Type-C und Gigabit Ethernet hergestellt.
Die Stereo-Verarbeitungskette ist vollständig in dem FPGA-Teil realisiert und enthält neben einem vorgestellten Sparse Retina Census Correlation (SRCC) Verfahren auch noch Module zur Vorverarbeitung, Stereo-Bildrektifizierung und Nachverarbeitung. Die gesamte Verarbeitung kann durch den Einsatz von Hardware-Modulen im FPGA mit sehr geringen Latenzen und damit in Echtzeit ausgeführt werden. Das Kamerasystem erreicht konstante Bildraten von 60 Bildern/s bei Bildauflösungen von 1280x720 Pixeln. Das Kamerasystem ist in Abbildung 3.14(a) gezeigt.



(a) (b)

Abbildung 3.14: (a) AeonCam smarte Stereo-Kamera mit FPGA-Verarbeitung (b) Einfluss der Pixel-Nachbarschaft auf die Census-Transformation [FA13]

Das in der Arbeit vorgestellte SRCC-Verfahren verwendet eine Kombination des "Sum of Absolute Differences" (SAD) Algorithmus und des "Sum of Hamming Distance" (SHD) Algorithmus zur Berechnung von Tiefeninformationen. Für die Verarbeitung wird dazu in einem Vorverarbeitungsschritt mit Sobel-Kantenfilter und einer Census-Transformation eine komprimierte Repräsentation der eingehenden Pixeldaten erstellt.
Eine Census Transformation nutzt die lokale $N \times N$ Umgebung zur Beschreibung in einem binären Wort. Für den mittleren p und einem Pixel p' aus der Nachbarschaft $W(p)$ mit der Intensität $I(p)$ erhält man die Census Transformation nach:

$$C(p) = \bigotimes_{p' \in W(p)} \xi(p, p') \tag{3.3}$$

mit

$$\xi(p, p') = \begin{cases} 1 & \text{if } I(p') < I(p) \\ 0 & \text{otherwise.} \end{cases} \tag{3.4}$$

Somit wird ein Bit in dem binären Wort gesetzt, wenn die Intensität eines Pixels aus der Nachbarschaft kleiner ist als die des mittleren Pixels. Die Differenz von zwei Census Strings kann über eine Hamming-Distanz berechnet werden, die in einer integrierten Schaltung sehr schnell durch eine XOR-Operation, Bitmasken und Addierern gebildet werden kann.
In der Arbeit wird ein Sparse Retina Census gezeigt, der nur die Pixel aus der Umgebung nutzt, die den meisten Einfluss auf die Korrelation hat. Die höchsten Korrelation-Relevanz haben die Pixel in einer kreisförmigen Umgebung der mittleren Pixels. Die Untersuchung der Korrelations-Eigenschaften ist in Abbildung 3.14(b) dargestellt. In dem SRCC-Algorithmus wird mit der Pixel-Umgebung aus einem 5x5 Fenster ein Sparce Census String mit 8 Bit generiert.
Aus den Sobel-Daten und Census-Daten lassen sich die Korrelationskosten als Summe der SAD- und SHD-Kosten über

$$C_{SAD} = \sum_{(x,y)\in W} |I_R(x,y) - I_L(x+d,y)| \tag{3.5}$$

und

$$C_{SHD} = \sum_{(x,y)\in W} Hamming(C_R(x,y), C_L(x+d,y)) \tag{3.6}$$

bestimmen.
Die Hardware-Implementierung verwendet ein 17x17 Akkumulationsfenster zur Berechnung der Korrelation. Das Verfahren nutzt Pipelining und parallele Einheiten zur gleichzeitigen Bestimmung von 256 Disparitäten.

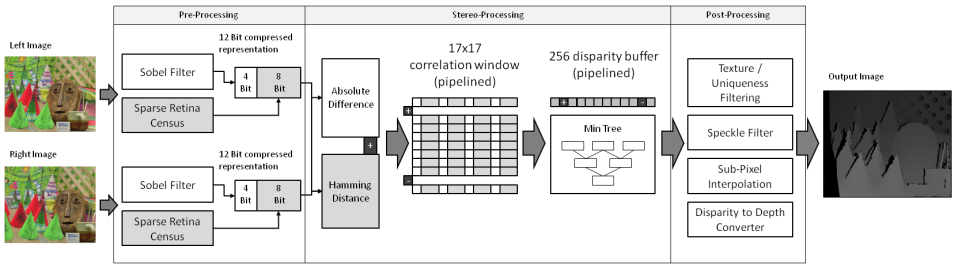


Abbildung 3.15: SRCC Stereo Pipeline

Das Verfahren implementiert zusätzlich eine Sub-Pixel Interpolation aus den besten und zweitbesten Korrelationswerten zur Sub-Pixel genauen Disparitäts- und Tiefenbestimmung. Abbildung 3.15 zeigt die SRCC Pipeline.
Der SRCC Algorithmus wurde mit dem Middlebury 2014 Dataset mit Auflösungen von bis zu 1500x1000 Pixeln evaluiert. Die erstellten Disparitätsbilder sind in Abbildung 3.16 dargestellt. In der Middlebury Evaluation erreicht

Name	Durchschnittlicher Fehler	Durchschnittliche Laufzeit
IDR	4.58%	0.34s
SRCC	5.41%	0.02s
TMAP	5.88%	1557s
SNCC	6.03%	0.97s
3DMST	7.08%	167s
INTS	7.42%	104s
SGM	7.52%	6.48s
APAP-Stereo	7.53%	117s
PMSC	8.20%	579s
Cens5	8.27%	1.34s

Tabelle 3.1: Middlebury Evalutation Top Ten - Durchschnittliche Fehler und durchschnittliche Laufzeiten, Half Resolution (bad pixel 2.0, nonocc)

das SRCC-Verfahren den 2. Platz der Rangliste mit einem durchschnittlichen Fehler von 5,41%. Dabei ist die Ausführungszeit 17x schneller als die zweitbeste GPU-basierte Implementierung.

Abbildung 3.16: SRCC Middlebury Ergebnisse: Links: linkes Bild. Rechts: Disparity-Bild

Aufgrund der reinen Hardware-Implementierung der SRCC Stereo-Tiefenberechnung kann das Modul ebenfalls in einem PCIe-basierten FPGA-Beschleuniger, wie er in dieser Dissertation als Ziel-Plattform verwendet wird, integriert werden. Somit könnte das SRCC Verfahren in Kombination mit der Merkmalsextraktion eingesetzt werden, um die Tiefenverarbeitung in SLAM-Systemen zu beschleunigen und zu verbessern.
Auch einzelne Teile der entwickelten Module sind für diese Dissertation interessant, da die Census Transformation Ähnlichkeiten zu binären Merkmalsdeskriptoren aufweist. Zum Deskriptor-Vergleich wird z.B. ebenfalls die Hamming Distanz verwendet.

Kapitel 4

Optimierung von merkmalsbasierten SLAM Systemen

In diesem Kapitel sollen die verschiedenen verfügbaren Algorithmen zur Merkmals-Erkennung hinsichtlich ihrer Eignung für die SLAM-Lokalisierung untersucht werden. Darüber hinaus werden in den Kapiteln 4.2 und 4.3 Verfahren zur Subpixel-genauen Bestimmung der Merkmalsposition und zur Erweiterung der Merkmals-Deskriptoren um Farbinformationen eingeführt und analysiert.

4.1 Auswahl der Algorithmen

In merkmalsbasierten SLAM-Systemen können falsche Zuordnungen von Landmarken zur Navigation oder falsche Korrespondenzen bei der Berechnung der Kamerabewegungen fatale Folgen für die Kartenerstellung haben. Die visuelle Odometrie eines SLAM-Systems ist dabei stets mit Drift behaftet. Der Drift kann nur durch eine gute Qualität der zu Grunde liegenden Algorithmen reduziert werden.
Ein Merkmalserkennungs-Algorithmus für merkmalsbasierte SLAM-Systeme benötigt daher folgende Eigenschaften:

- kurze Berechnungszeit / schnelle Bildraten für den Einsatz in visueller Odometrie
- Robustheit gegen Variationen aus Bildaufnahmen mit verschiedenen Blickrichtungen, Skalierungen und Beleuchtungen
- schnelles und eindeutiges Matching von Deskriptoren
- geringer Ressourcenverbrauch und Leistungsaufnahme für den Einsatz in mobilen Systemen

Dabei kann die Rotations- und Skalen-Invarianz für die Odometrie-Berechnung aufgrund der nah beieinander liegenden Bildaufnahmen geringer sein als z.B. für die Erkennung von bereits aufgedeckten Kartenbereichen (Loop-Closures).
Idealerweise kann aber ein Detektor/Deskriptor-System genutzt werden, das für beide Anwendungen gleichermaßen geeignet ist, um die Merkmale wiederzuverwenden und somit Berechnungszeit zu sparen.
In [MTS+05] wurde das Oxford Dataset zur Evaluation von Merkmals-Erkennungen vorgestellt. Die darin enthaltenen Bilder enthalten jeweils sechs verschiedene Bildaufnahmen einer Szene zur Analyse wichtiger Eigenschaften von Merkmals-Detektoren und Deskriptoren. Abbildung 4.1 zeigt alle Bildsequenzen des Oxford Datasets. Das Dataset enthält zusätzlich die Homographien der Einzelbilder zum Ausgangsbild als Ground Truth Referenz der jeweiligen Bildtransformation.

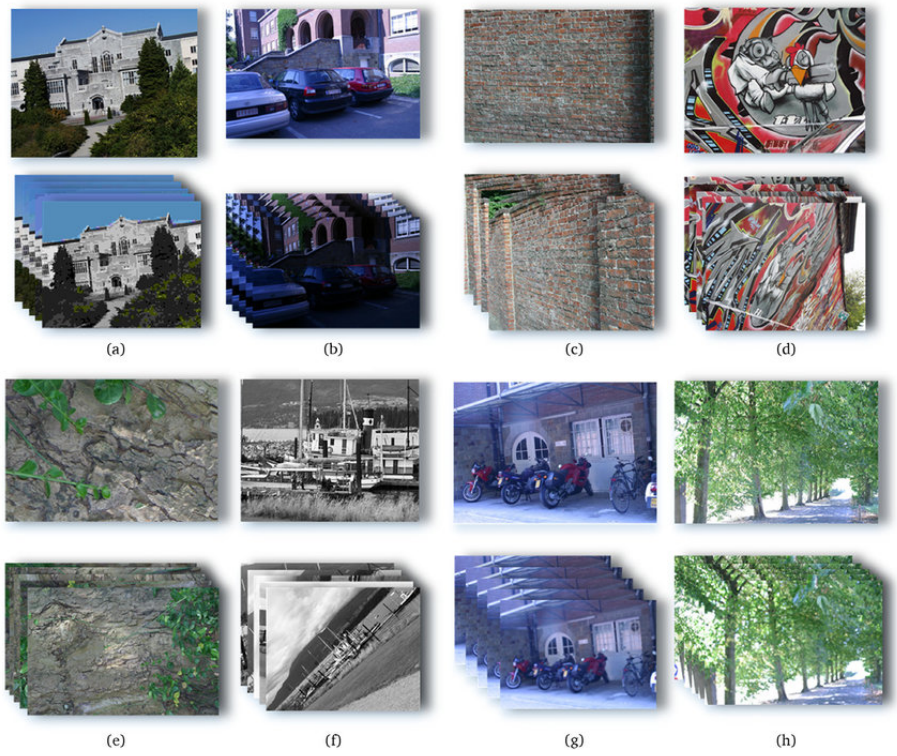


Abbildung 4.1: Oxford Affine Covariant Feature Dataset - (a) Compression (b) Light (c)/(d) Viewpoint (e)/(f) Zoom+Rotation (g)/(h) Blur [MTS+05][YDMJ18]

Für diese Dissertation wurde mit dem Oxford-Dataset eine Evaluation aller verfügbaren und für SLAM-Systeme interessanten Algorithmen durchgeführt, um sie hinsichtlich ihrer Eignung zu untersuchen.

Detektor/ Deskriptor	∅ Matches	Inlier H1	Inlier H2	Inlier H3	∅Det. Zeit	∅Deskr. Zeit
SIFT/SIFT	468	98.9%	94.8%	90.7%	187ms	122ms
SURF/SURF	377	97.3%	94.8%	88.7%	119ms	184ms
AKAZE/AKAZE	573	99.3%	97.8%	92.1%	117ms	106ms
BRISK/BRISK	331	98.7%	83.7%	86.6%	31ms	21ms
ORB/ORB	470	98.7%	96.5%	97.0%	15ms	12ms
ORB/FREAK	319	98.4%	96.8%	96.7%	15ms	47ms
ORB/LATCH	300	99.5%	72.4%	55.5%	15ms	155ms

Tabelle 4.1: Evaluation der Merkmals-Algorithmen, Oxford Datasets H1-H3, 2000 Merkmale, Intel I7-6700(3,4GHz)

Dazu wurden die OpenCV-3.2 Implementierungen der Algorithmen auf einem Intel Core i7 Prozessorsystem mit 3,4GHz Prozessortakt ausgeführt und gleichzeitig die Ausführungszeiten gemessen. Im Anschluss wurden die Matching Ergebnisse aus dem Vergleich der jeweils erstellten Deskriptoren bestimmt.
Tabelle 4.1 zeigt die Ergebnisse der verschiedenen Algorithmen. Über die Parameter der Algorithmen wurde die Anzahl der Merkmale auf 2000 eingestellt, um einen besseren Vergleich der Laufzeiten zu ermöglichen. Zur Evaluation wurden die durchschnittliche Anzahl an Matches und das Inlier Verhältnis (4px Variation und 0.6 Nearest-Neighbour-Verhältnis) über die ersten 3 Bilderpaare/Homographien aller Datasets bestimmt. Die Spalten Detektor-Zeit und Deskriptor-Zeit zeigen die durchschnittliche Laufzeit über alle 24 Bilder.
Die gemessenen Daten zeigen, dass der ORB-Algorithmus eine hohe Anzahl an Matches und eine hohe Inlier-Rate bei der geringsten durchschnittlichen Laufzeit bietet. Dieser Algorithmus wurde auch schon in anderen bestehenden SLAM Systemen z.B. ORB-SLAM und RTAB-Map für die Odometrieberechnung und die Erstellung der Landmarken verwendet.
Aufgrund der guten Ergebnisse und der vergleichsweise einfachen Implementierung eignet sich der ORB-Algorithmus für Lokalisierungssysteme und die Umsetzung auf parallelen Architekturen. Auch die geringere Matching-Zeit durch den binären ORB-Deskriptor ist ein entscheidener Vorteil für merkmalsbasierte SLAM-Implementierungen.
Daher wurde für diese Dissertation der ORB-Detektor und Deskriptor als Basis für den zu entwerfenden Algorithmus ausgewählt.

4.2 Subpixel-Interpolation von Merkmals-Koordinaten

Merkmals-Koordinaten mit Subpixel-Genauigkeit bieten eine erhöhte Präzision für Lokalisierungs-Berechnungen im Vergleich zu herkömmlichen Merkmals-Koordinaten auf Pixel-Raster, da in einem Bild ein Merkmal oder eine Objektkante nicht immer auf dem verwendeten Raster liegt. Um eine Subpixel-genaue Bildposition zu erhalten, kann dazu über die Kantenantworten der Merkmalsumgebung eine Subpixel Interpolation in der Nachverarbeitung ausgeführt werden. Ein Beispiel eines Merkmalspunktes mit Subpixel-Genauigkeit ist in Abbildung 4.2 gezeigt.

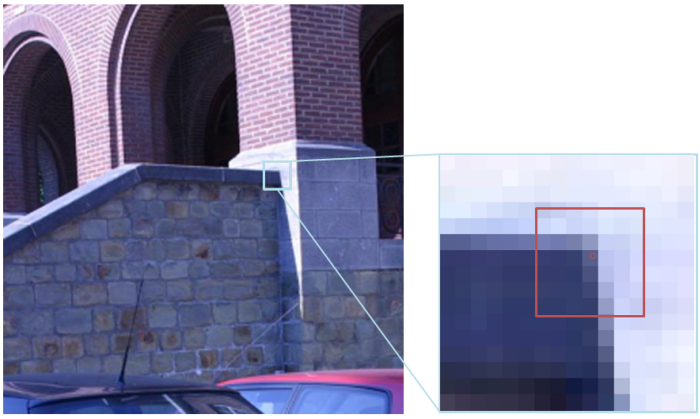


Abbildung 4.2: Subpixel Interpolation der Merkmals-Koordinaten

Feature Detektoren von Algorithmen wie SIFT, BRISK und AKAZE nutzen diese Informationen, um die Merkmals-Koordinaten mit erhöhter Genauigkeit über den gesamten Skalenraum zu erhalten und zu interpolieren. Allerdings sind die meisten der Algorithmen mit integrierter Subpixel-Interpolation der Merkmals-Koordinaten aufgrund ihrer komplexen Konstruktion und des verwendeten Skalenraums sehr rechenaufwändig. Selbst die auf Ausführungszeit optimierten Verfahren mit binären Deskriptoren wie BRISK und AKAZE, die Subpixel-Informationen nutzen, sind durch ihre Berechnungsdauer immer noch ungeeignet für die Echtzeit-Anforderungen eines visuellen Lokalisierungssystems. Die nächsten Kapitel sollen eine Übersicht der verwendeten Subpixel-Interpolationsverfahren bieten, um eine geeignete Implementierung für die Anforderungen in Detektoren für ein SLAM-System mit Hardware-Beschleuniger auszuwählen.

Subpixel Interpolation durch Lösung von linearen Gleichungssystemen

Der SIFT Merkmals-Detektor [Low04] benutzt die Methode von Brown [BL02] zur Anpassung einer 3D quadratischen Funktion zur Subpixel-genauen

Positionsbestimmung aus der lokalen Nachbarschaft des Merkmals. Dadurch konnten Verbesserungen des Matchings und der Stabilität erreicht werden. Die Methode benutzt die Taylor-Reihen Erweiterung der auf den Merkmalspunkt verschobenen Skalenraums-Funktion $D(x, y, \sigma)$:

$$D(x) = D + \frac{\partial D^T}{\partial x}x + \frac{1}{2}x^T\frac{\partial^2 D}{\partial x^2}x \tag{4.1}$$

Dabei ist $x = (x, y, \sigma)^T$ der Abstand von dem Merkmalspunkt. Die Lage des Maximums $\hat{x}$ kann durch Ableitung nach x und durch Null setzen berechnet werden:

$$\hat{x} = -\frac{\partial^2 D^{-1}}{\partial x^2}\frac{\partial D}{\partial x} \tag{4.2}$$

Nach Brown [BL02] können die Hessesche Matrix H und die Ableitungen von D über die Differenzen der benachbarten Punkte angenähert werden. Dadurch entsteht ein leichter zu lösendes 3x3 Gleichungssystem der Form

$$HX = D. \tag{4.3}$$

In AKAZE [AS11] wird die Subpixel-genaue Position des Merkmals über die Bestimmung des Maximums mit Anpassung einer 2D quadratischen Funktion zur Determinante der Hesseschen-Antwort in einer 3x3 Pixel Nachbarschaft berechnet. Dazu werden die Abstände dx, dy über die Gradienten Dx, Dy und die Hesseschen Werte Dxx, Dyy, Dxy durch Lösen des Gleichungssystems

$$\begin{pmatrix} Dxx & Dxy \\ Dxy & Dyy \end{pmatrix} * \begin{pmatrix} dx \\ dy \end{pmatrix} = \begin{pmatrix} -Dx \\ -Dy \end{pmatrix} \tag{4.4}$$

bestimmt.

Subpixel Interpolation über die Methode der kleinsten Quadrate mit quadratischem Gleichungssystem

In [LCS11] verwendet der Autor zur Subpixel-Interpolation ein quadratisches Polynom mit $k = 6$ Parametern der Form

$$f(x, y) = a_0x^2 + a_1y^2 + a_2xy + a_3x + a_4y + a_5 \tag{4.5}$$

um eine paraboloide Fläche anzupassen. Die Parameter $a_0...a_5$ können über die Methode der kleinsten Quadrate berechnet werden.
Die Subpixel-genauen Merkmals-Koordinaten $\hat{x}, \hat{y}$ erhält man dann über

$$\begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} k_4k_5 + \dfrac{2k_2k_3}{4k_1k_2 - {k_5}^2} \\ k_3k_5 + \dfrac{2k_1k_4}{4k_1k_2 - {k_5}^2} \end{pmatrix}. \tag{4.6}$$

Der BRISK Algorithmus verwendet zusätzlich noch eine 1D Parabelanpassung über die Skalen-Achse, um die Koordinaten auch über den Skalenraum zu optimieren. Der Skalenraum und die Koordinaten-Interpolation sind in Grafik 4.3 gezeigt.

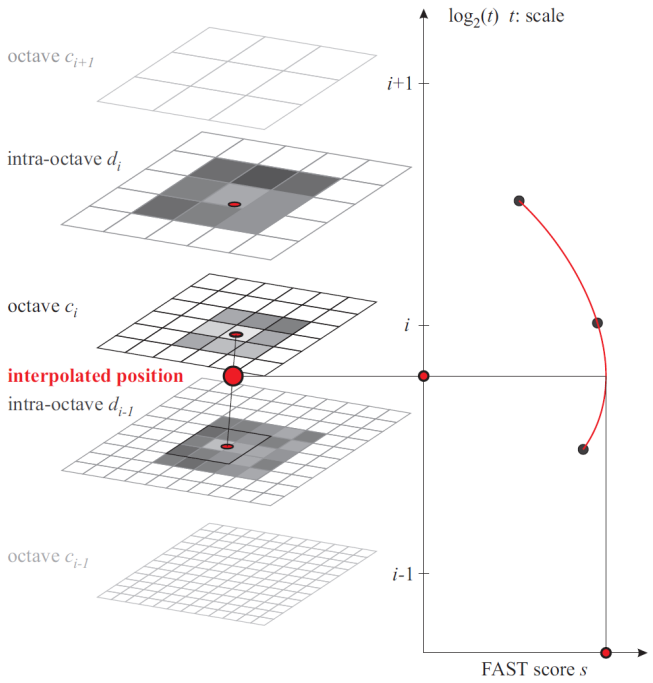


Abbildung 4.3: BRISK Skalenraum mit Subpixel-Interpolation [LCS11]

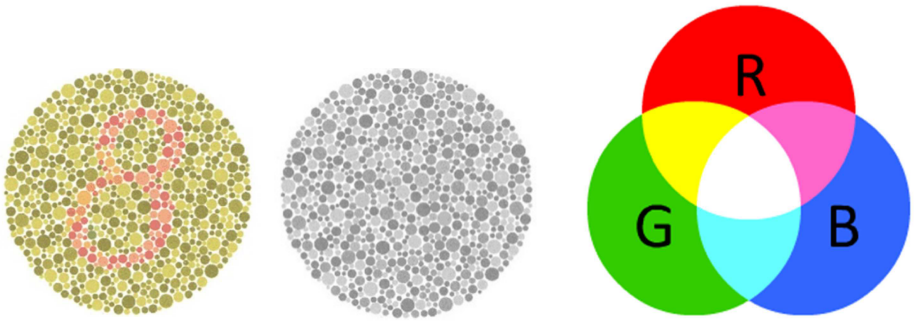
Die Subpixel-Interpolationsverfahren wurden in Kombination mit dem FAST Detektor und Bildern aus dem Oxford Datensatz gegeneinander getestet und hinsichtlich der Auswirkungen auf die Ergebnisse, ihrer Komplexität und der Eignung für eine Hardware-Implementierung untersucht. Tabelle 4.2 fasst die Ergebnisse zusammen. Dabei konnten mit der Methode der kleinsten Quadrate mit quadratischem Gleichungssystem, die auch in dem BRISK Detektor eingesetzt wird, die besten Ergebnisse bei einer geringen Komplexität erzielt werden.

	SIFT	AKAZE	BRISK
Abhängigkeit vom Skalenraum	–	+	+
Ergebnisse	++	+	++
Komplexität	–	-	+
HW-Implementierbarkeit	–	+	++

Tabelle 4.2: Eigenschaften der Subpixel-Interpolationsverfahren

4.3 Bildmerkmale mit Farbinformationen

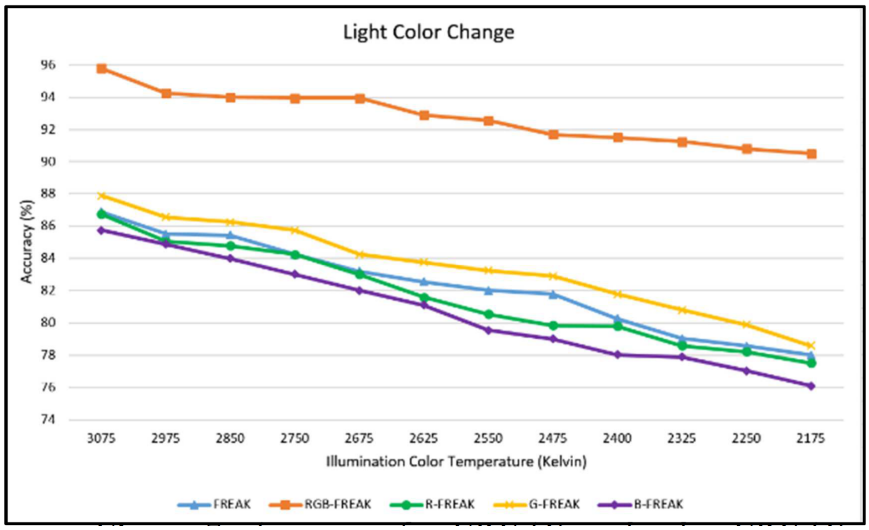
Herkömmliche Merkmals-Detektoren und Deskriptoren verwenden Graustufenbilder zur Erkennung und Beschreibung von Merkmalen. Dabei hat jedes Pixel einen Intensitätswert mit einer Bit-Tiefe von üblicherweise 8-16Bit. Farbinformationen können zusätzliche Informationen einer Bildszene liefern.



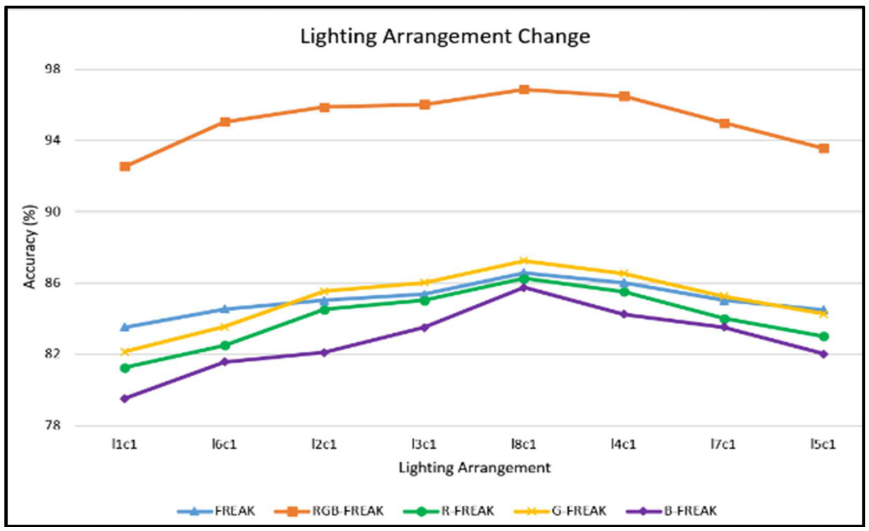
(a) (b)

Abbildung 4.4: (a)Darstellung eines Bildmusters als Farbbild und Graustufenbild [TAA16] (b) RGB-Farbraum [TAA16]

In Abbildung 4.4 a) ist ein Bildmuster dargestellt, das nur in Farbbildern sichtbar ist. Daher kann hier durch Verwendung mehrerer Farbkanäle eine bessere Unterscheidung getroffen werden. Zur Darstellung von Farbbildern werden in der Bildverarbeitung verschiedene Farbräume verwendet. In dem RGB-Farbraum werden dazu die Farbinformationen eines Bildes in die drei Farbkanäle Rot, Grün und Blau aufgeteilt. Das RGB-Modell ist in Abbildung 4.4 b) dargestellt. In [TAA16] wurde der binäre FREAK Deskriptor um Farbinformationen erweitert. Dazu wurde für jedes Merkmal der Deskriptor für alle drei Farbkanäle separat berechnet und am Ende ein RGB-Deskriptor durch Konkatenation der drei Farb-Deskriptoren erstellt. Der RGB-FREAK Algorithmus wurde in der Arbeit mit dem ALOI Dataset mit Bildaufnahmen von 1000 Objekten mit unterschiedlicher Beleuchtung und Farbtemperatur getestet. Die Abbildungen 4.5 a) und 4.5 b) zeigen die berechnete Genauigkeit aus der Anzahl der korrekten Matches und der Anzahl der gefundenen Merkmale über die Variation der Beleuchtung und der Farbtemperatur.
Die Ergebnisse zeigen, dass für den ALOI Datensatz durch die Verwendung der RGB-Kanäle zur Deskriptorerstellung ein Vorteil gegenüber der FREAK-Implementierung mit Graustufenbildern oder FREAK-Varianten mit einzelnen Farb-Komponenten erreicht werden kann. Der Nachteil der Implementierung mit RGB-Deskriptor ist nach Aussagen der Autoren die höhere Berechnungszeit, die im Vergleich zum originalen FREAK Deskriptor fast drei mal höher ist.
In [Bra13] wurde ein ähnliches Verfahren angewandt, um den FREAK Al-



(a)



(b)

Abbildung 4.5: Genauigkeit des FREAK und Color-FREAK Algorithmus: (a) Variation der Beleuchtungskonfigurationen (b) Variation der Farbtemperaturen [TAA16]

gorithmus mit Farbinformationen zu erweitern. Die Arbeit hat zusätzlich auch andere Farbräume und Farbdarstellungen analysiert. So wurden neben dem RGB-Format auch Farbdarstellungen mit Gegenfarben (oppFREAK) und Gegenfarben mit Intensität (oppiFREAK) sowie der HSV-Farbraum mit Farbton-, Sättigungs- und Helligkeitskanälen untersucht.

Zur Evaluation wurde das Oxford Dataset [MTS+05] verwendet, um die

Recall/1-Precision Kurven der Deskriptor-Implementierungen zu bestimmen. Die Recall/1-Precision Kurven der Deskriptor-Implementierungen für die Datensätze Bark, Bikes, Boat und Grafitti sind in Abbildung 4.7 gezeigt. Die Flächen unter den Recall/1-Precision Kurven der Oxford Datensätze für alle FREAK Implementierungen ist in Grafik 4.6 dargestellt.

	FREAK	rgbF.	oppF.	oppiF.	hueF.	hsvF.	rgbF./F.
bark	0.395	0.436	0.148	0.339	0.112	0.375	1.104
bikes	0.791	0.793	0.469	0.666	0.358	0.698	1.003
boat	0.611	0.611	0.0	0.609	0.0	0.609	1.000
graf	0.368	0.437	0.171	0.286	0.073	0.323	1.188
leuven	0.756	0.744	0.351	0.532	0.312	0.524	0.984
trees	0.420	0.435	0.283	0.382	0.130	0.386	1.036
ubc	0.905	0.887	0.029	0.405	0.009	0.333	0.980
wall	0.511	0.584	0.527	0.638	0.336	0.636	1.143

Abbildung 4.6: FREAK Deskriptor mit verschiedenen Farbdarstellungen, Fläche unter den Recall/1-Precision Kurven der Oxford Datasets [Bra13]

Hier bietet ebenfalls die Implementierung mit dem RGB-Farbmodell die besten Ergebnisse. Allerdings stellt der Autor eine Abhängigkeit von dem verwendeten Datensatz fest. So kann in den Datensätzen Leuven und UBC im Vergleich zu der normalen FREAK Implementierung keine Verbesserung erreicht werden. Die Betrachtung von Farbe kann sich bei Beleuchtungsänderungen und schlechter Bildqualität zum Nachteil entwickeln, da die Merkmals-Deskriptoren durch die Farbinformation weniger robust gegenüber diesen Variationen sind.
Für diese Dissertation wurde ebenfalls eine Analyse mit verschiedenen Farbräumen und Farbdarstellungen durchgeführt. So wurden auch verschiedene Konfigurationen des ORB-Deskriptors mit RGB-, HSV- und YUV-Farbmodellen sowie Histogramm-Darstellungen untersucht. Hier konnte eine ähnliche Abhängigkeit beobachtet werden. Der RGB-Farbraum liefert aufgrund seiner guten kontrastreichen Repräsentation der Farbkanäle die besten Ergebnisse.

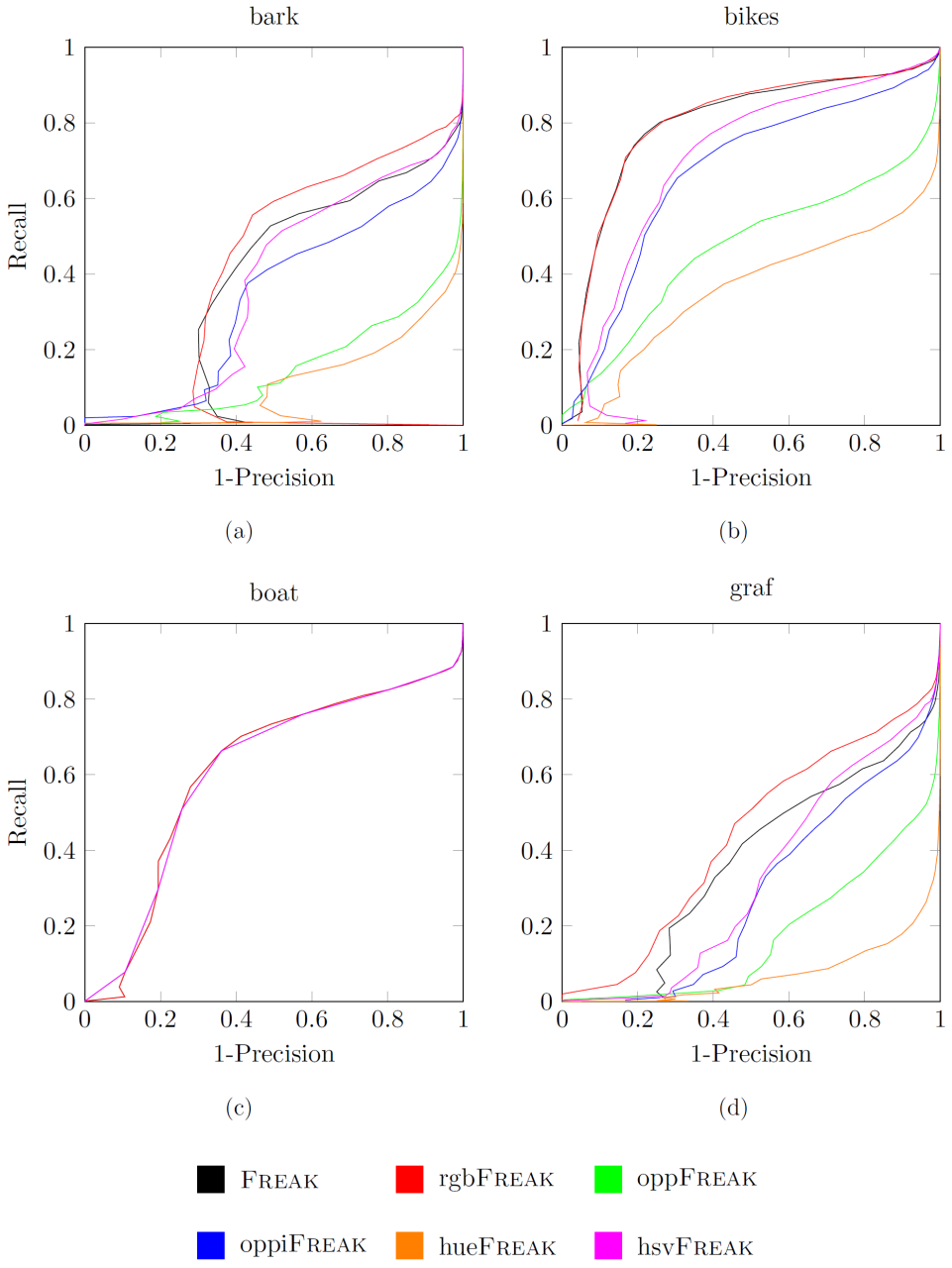


Abbildung 4.7: FREAK Deskriptor mit verschiedenen Farbdarstellungen, Recall/1-Precision Kurven der Datasets Bark, Bikes, Boat und Grafitti [Bra13]

Kapitel 5

Entwurf einer Merkmalserkennung für SLAM Systeme

In diesem Kapitel soll eine für SLAM Systeme entwickelte Subpixel-genaue Merkmalserkennung vorgestellt und die einzelnen Bestandteile im Detail betrachtet werden. Das Kapitel 5.1 stellt dabei den, in dieser Arbeit entworfenen, Subpixel-accurate Oriented AGAST and Rotated BRIEF (SOARB) Algorithmus vor. In Kapitel 5.2 wird eine Version des Algorithmus' mit einer zusätzlichen Erweiterung des Deskriptors mit Farbinformationen gezeigt.

5.1 SOARB - Subpixel-accurate Oriented AGAST and Rotated BRIEF

Für diese Arbeit wurde eine für SLAM-Algorithmen optimierte Version des ORB-Algorithmus entwickelt. Der vorgestellte Subpixel accurate Oriented AGAST and Rotated BRIEF (SOARB) verwendet dabei einen erweiterten AGAST Detektor mit Subpixel genauer Bestimmung der Merkmals-Koordinaten. Durch die Subpixel-Interpolation können die Merkmals-Koordinaten im 3D-Raum bessere Informationen für die Visuelle Odometrie und die Lokalisierung liefern.
Die Rotationsinvarianz und Skaleninvarianz wird dabei, wie auch im ORB Detektor, durch die Bestimmung der Merkmals-Rotation über einen Intensitäts-Zentroiden und die Verwendung von Skalenpyramiden realisiert. Zu Beginn der Merkmalsextraktion wird dazu die Skalenpyramide durch Skalieren und Unterabtasten des Eingangsbilds erstellt. Der Aufbau einer SOARB Skalenpyramide mit 4 Leveln ist in Grafik 5.1 dargestellt. Zur Implementierung und Evaluation des SOARB Algorithmus wurden Skalenpyramiden mit 8 Leveln und einem Skalierungsfaktor von 1,2 eingesetzt.

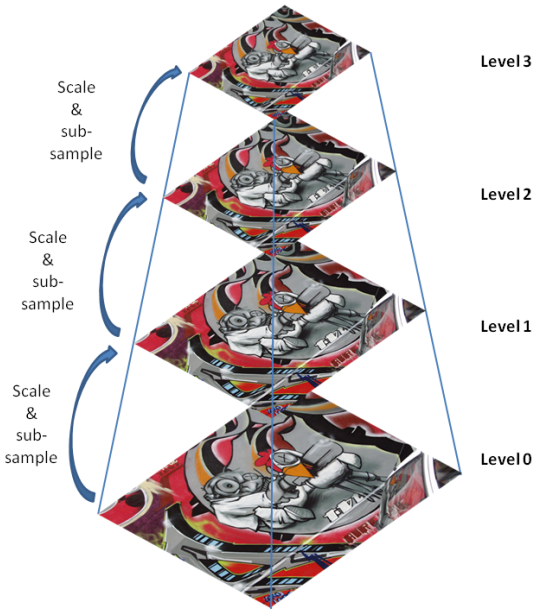


Abbildung 5.1: SOARB Skalen-Pyramide für 4 Level

Der SOARB Deskriptor ist ein binärer Merkmalsdeskriptor der auf dem ORB Deskriptor-Pattern mit 256 Vergleichs-Paaren basiert. Das verwendete Deskriptor-Pattern ist in Abbildung 5.2 gezeigt.

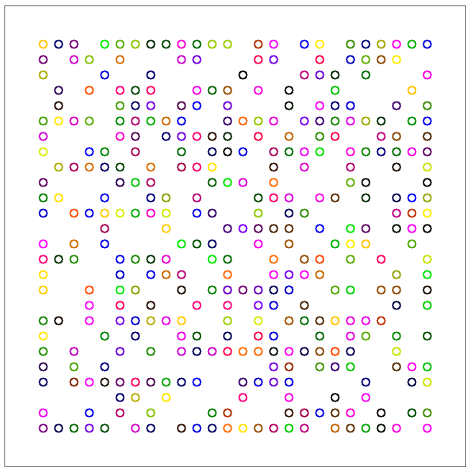


Abbildung 5.2: SOARB Deskriptor Pattern

Die Pattern-Punkte werden dabei, wie auch beim ORB-Verfahren, bei der Deskriptor-Generierung um den Orientierungswinkel gedreht. Dabei wurde zusätzlich eine Subpixel-genaue Interpolation der Pixelwerte nach der Rota-

tion der Pattern-Punkte eingeführt, um eine verbesserte Differenzierung von Drehungen um die Bildachse zu ermöglichen. Dadurch wird der Detektor robuster gegenüber Kamera-Rotationen. Abbildung 5.3 zeigt die Rotation des Deskriptor-Pattens um die Merkmals-Orientierung.

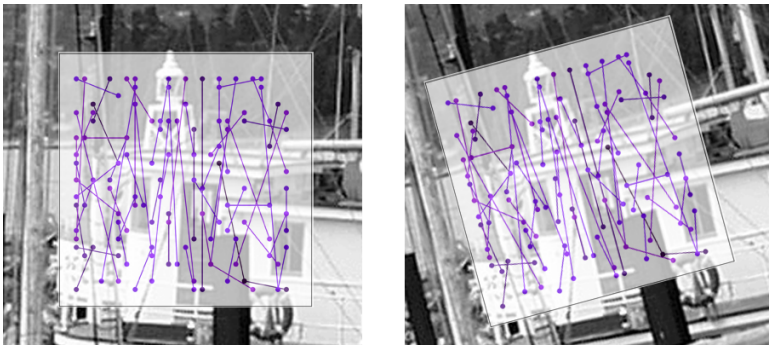
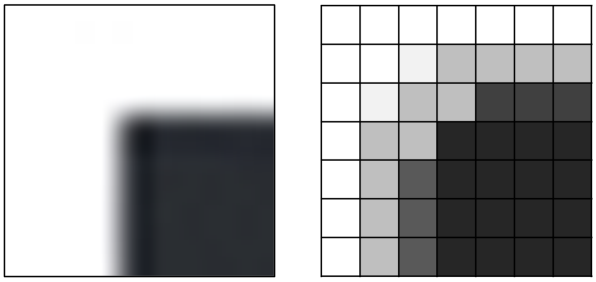


Abbildung 5.3: SOARB Deskriptor Pattern Orientation

5.1.1 Subpixel Interpolation der Merkmals-Koordinaten

Um eine Subpixel-genaue Bildposition der Merkmale zu erhalten, wurde für den SOARB-Algorithmus eine Interpolation über die Methode der kleinsten Quadrate mit quadratischem Gleichungssystem nach [LCS11] implementiert. Es werden dazu die AGAST-Scores in einer 3x3 Nachbarschaft ausgewertet und über Kurvenanpassung eines Polynoms die Lage des Maximums interpoliert. Ein Beispiel mit einem simulierten Kantenbild und den entsprechenden AGAST-Scores ist in Abbildung 5.4 gezeigt.



0	0	0	0	0	0	0
0	0	11	26	26	0	0
0	0	101	183	153	47	0
0	1	128	197	84	31	1
0	0	101	88	26	0	0
0	0	0	31	0	0	0
0	0	0	0	0	0	0

(a) (b) (c)

Abbildung 5.4: (a) Eingangsbild (b) 7x7 Pixel Bildbereich (c) AGAST-Scores mit 3x3 Subpixel-Fenster

Aus den Scores $s_{0,0} - s_{2,2}$, wie in Abbildung 5.5 gezeigt, werden die Koeffizienten $k_1 - k_5$ des quadratischen Gleichungssystems nach Formel 5.1 berechnet.

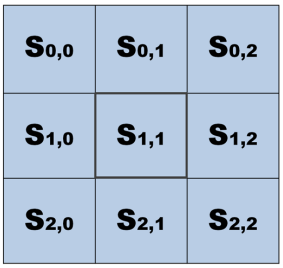


Abbildung 5.5: 3x3 Score-Fenster zur Subpixel-Interpolation

$$
\begin{aligned}
k_1 &= 3(s_{0,0} + s_{0,2} - 2s_{1,1} + s_{2,0} + s_{2,2} + s_{0,1} - 2(s_{1,0} + s_{1,2}) + s_{2,1}) \\
k_2 &= 3(s_{0,0} + s_{0,2} - 2s_{1,1} + s_{2,0} + s_{2,2} + s_{1,0} - 2(s_{0,1} + s_{2,1}) + s_{1,2}) \\
k_3 &= 3(s_{0,0} + s_{0,2} + s_{0,1} - s_{2,0} - s_{2,2} - s_{2,1}) \\
k_4 &= 3(s_{0,0} + s_{0,2} + s_{1,0} - s_{2,0} - s_{2,2} - 2(s_{0,2} - s_{2,0}) - s_{1,2}) \\
k_5 &= 4(s_{0,0} - s_{0,2} - s_{2,0} + s_{2,2})
\end{aligned}
\tag{5.1}
$$

Danach kann die Determinante nach

$$H_{det} = 4k_1k_2 - {k_5}^2 \tag{5.2}$$

bestimmt werden.
Über die Subpixel Abstände x' und y' mit

$$
\begin{aligned}
x' &= k_4k_5 + \frac{2k_2k_3}{H_{det}} \\
y' &= k_3k_5 + \frac{2k_1k_4}{H_{det}}
\end{aligned}
\tag{5.3}
$$

lassen sich die Subpixel-genauen Bildkoordinaten x_{sub} und y_{sub} mit Hilfe der ursprünglichen Bildkoordinaten x und y nach

$$
\begin{aligned}
x_{sub} &= x + x' \\
y_{sub} &= y + y'
\end{aligned}
\tag{5.4}
$$

berechnen. Die Interpolation konnte gegenüber [LCS11] vereinfacht werden, da der SOARB-Algorithmus einen einfacheren Aufbau des Skalenraums benutzt und daher im Anschluss keine 1D-Parabel-Optimierung der Skalen-Achse durchführt. Zusätzlich werden für den SOARB-Detektor lediglich die interpolierten Koordinaten des Merkmals benötigt und nicht der interpolierte Intensitätswert. Dadurch konnte der Implementierungsaufwand weiter reduziert werden.

5.1.2 Subpixel Interpolation des Deskriptor-Patterns

Der SOARB-Algorithmus verwendet den 256-Bit ORB-Deskriptor Aufbau aus Abbildung 5.2 und erweitert den Deskriptor zusätzlich um eine Subpixel-genaue Rotation des Deskriptor-Patterns mit bilinearer Interpolation der Pixelwerte.
Für eine Merkmals-Orientierung θ und die Pattern-Koordinaten p_x, p_y können die rotierten Pattern Koordinaten $p_{x'}, p_{y'}$ über

$$\begin{aligned} p_{x'} &= p_x * cos(\theta) - p_y * sin(\theta) \\ p_{y'} &= p_x * sin(\theta) + p_y * cos(\theta) \end{aligned} \tag{5.5}$$

berechnet werden.
Die Subpixel-Abstände rx, ry jeder Koordinate werden über

$$\begin{aligned} rx &= p_{x'} - floor(p_{x'}) \\ ry &= p_{y'} - floor(p_{y'}) \end{aligned} \tag{5.6}$$

bestimmt. Daraus lassen sich die Gewichte a_{00} bis a_{11} für die bilineare Interpolation berechnen:

$$\begin{aligned} a_{00} &= (1 - rx) * (1 - ry) \\ a_{01} &= rx * (1 - ry) \\ a_{10} &= ry * (1 - rx) \\ a_{11} &= rx * ry \end{aligned} \tag{5.7}$$

Die Intensität $I_{x',y'}$ des rotierten Pattern-Punktes $P_{x',y'}$ kann nun aus den Nachbarschafts-Intensitäten I_{00} bis I_{11} nach Formel 5.8 bestimmt werden.

$$I_{x',y'} = I_{00} * a_{00} + I_{01} * a_{01} + I_{10} * a_{10} + I_{11} * a_{11} \tag{5.8}$$

5.2 SOARB-RGB - Erweiterung des Deskriptors mit Farbinformationen

Zur Evaluation in Datasets mit Farbinformationen wurde ein angepasster Deskriptor entworfen. Dazu wurden verschiedene Deskriptor-Konstruktionen mit unterschiedlichen Farbrepräsentationen getestet.
In den Tests mit dem Oxford Dataset und KITTI-Dataset (siehe Kapitel 8.3.3) erzielte ein angepasster FREAK Deskriptor mit RGB-Farbinformationen die besten Ergebnisse. Abbildung 5.6 zeigt das Deskriptor Pattern des SOARB-RGB Deskriptors.
Das symmetrische FREAK-Pattern erlaubt eine bessere Gewichtung und Verteilung der Vergleichspaare über die Farbkanäle. Dabei wird ein 512Bit Deskriptor aus alternierenden Rot-, Grün- und Blau-Werten aufgebaut. Die

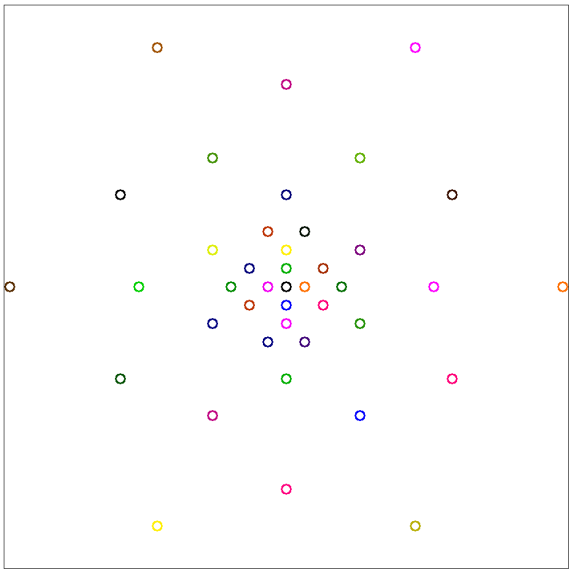


Abbildung 5.6: SOARB-RGB Deskriptor Pattern

grünen Pixel werden dabei doppelt gewichtet, indem 256Bit des Deskriptors dem Grün-Kanal entsprechen und jeweils 128Bit den Rot- und Blau-Kanälen.
Die Farbe Grün leistet beim menschlichen Auge den größten Beitrag zur Helligkeits- und Kontrastwahrnehmung. Das gleiche Prinzip der höheren Gewichtung der Grün-Anteile wird z.B. auch beim Aufbau eines Bildsensors mit Bayer-Pattern eingesetzt.

Abbildung 5.7: Anteile der Farbkanäle im SOARB-RGB Deskriptor

Der Deskriptoraufbau ist damit kompakter als die Konkatenation der drei Farb-Deskriptoren [TAA16] [Bra13]. Somit ist mit dem SOARB-RGB Deskriptor eine geringere Ressourcenanforderung und geringere Matching-Zeit beim Deskriptorvergleich zu erwarten.

Kapitel 6

Hardware Design

In diesem Kapitel wird in den Sektionen 6.1 und 6.2 die zum Hardware Design verwendete Entwicklungsumgebung mit OpenCL Unterstützung gezeigt. Die GPU- und FPGA-Implementierungen des SOARB Algorithmus werden in den Sektionen 6.3 und 6.4 im Detail vorgestellt.

6.1 OpenCL

OpenCL ist ein Programmiersprachen-Standard für parallele Recheneinheiten, der aus der Zusammenarbeit von AMD, IBM, Intel, Nvidia und Apple entstanden ist. OpenCL bietet dabei ein Plattform-unabhängiges, offenes Programmiermodell und hat im Vergleich zu proprietären Lösungen, wie z.b. Nvidia CUDA, eine breite Unterstützung von Produkten verschiedener Hersteller sowie verschiedener Geräte-Architekturen.
Ein OpenCL-System besteht aus einem Host-Prozessor und einem oder mehreren OpenCL-Geräten. Ein Gerät verfügt dabei über einzelne oder mehrere unabhängige Recheneinheiten. Das können z.B. Shader-Kerne einer Grafikkarte, Hardware-Elemente eines FPGA-Beschleunigers oder einzelne Prozessorkerne eines Multi-Prozessors sein. Abbildung 6.1 zeigt das OpenCL Plattform Modell.
In einem OpenCL-System können die Anwendungen (Kernel) vom Host-Prozessor zur Laufzeit auf den verfügbaren Geräten ausgeführt werden. Der Host kommuniziert mit den Geräten über den PCIe-Systembus und kann durch Verwendung der OpenCL API-Funktionen folgende Aufgaben erfüllen:

- Verwaltung des Betriebssystems und der Gerätetreiber
- Ausführen der Host-Anwendung
- Aufsetzen von globalen Speichern und Verwaltung des Datenaustauschs zwischen Host und Devices.
- Laden und Ausführen der Kernel-Programme auf den Devices

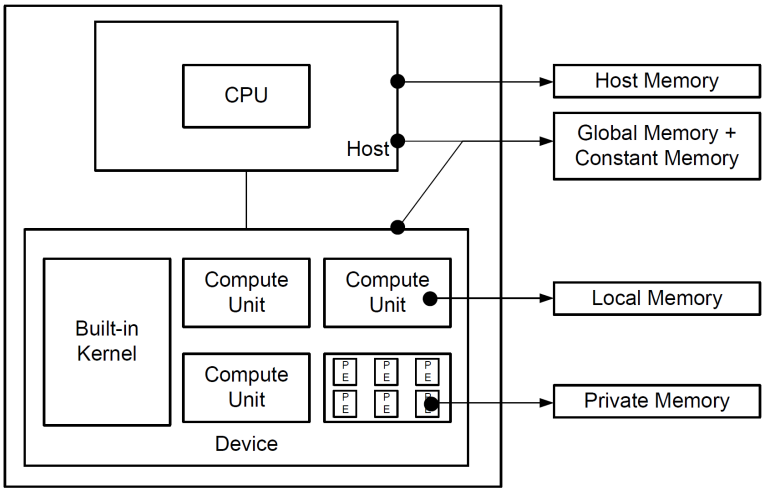


Abbildung 6.1: OpenCL Platform- und Speichermodell [xild]

Der Datenaustausch zwischen Host und Geräten ist durch Zeiger zu globalen Speicherbereichen realisiert, zusätzlich können Kernel-Parameter bei der Ausführung übergeben werden.
Seit der OpenCL 2.0 Spezifikation unterstützt der Standard ebenfalls FIFO-Speicherelemente für Streaming-Anwendungen mit den „Pipe“-Elementen. Dadurch lassen sich sehr geringe Verzögerungszeiten bei der Kommunikation zwischen Kerneln und einzelnen Kernelfunktionen erreichen. Diese Elemente lassen sich ebenfalls in OpenCL FPGA-Beschleunigern nutzen, um die für die FPGA-Architekturen üblichen Streaming-Methoden zu implementieren.
In OpenCL wird jeder Kernel mit einer lokalen und globalen Workgroup-Size ausgeführt, somit lassen sich die Anzahl der parallelen Berechnungen (Threads) konfigurieren und die Berechnungen in Gruppen unterteilen.
Dieser generische Ansatz erlaubt es, eine Vielzahl von unterschiedlichen Geräte-Architekturen mit einer beliebigen Anzahl an Prozessoren, Recheneinheiten und Speicherkonfiguration zu unterstützen.
Da die OpenCL Geräte-Codes entweder während der Kompilierung des Host-Codes mit übersetzt werden oder auch als vor-kompilierte binäre Dateien eingelesen werden können, eröffnet das Framework neue Möglichkeiten zur Integration von weiteren Beschleunigern-Architekturen in Hardware-Software Anwendungen. Bei FPGA-Architekturen mit Unterstützung von partieller Rekonfiguration kann somit das Bit-File mit den entsprechenden Kernel-Modulen über das OpenCL-System zur Laufzeit geladen werden.
Damit bietet OpenCL ein ideales Werkzeug für den Entwurf von FPGA-Beschleuniger Systemen mit hoher Datenparallelität und für die in dieser Dissertation gestellten Anforderungen.

6.2 Xilinx SDAccel Umgebung

Die Xilinx SDAccel-Umgebung ist ein Entwicklungs-Tool mit integrierter High-Level-Synthese für FPGA-basierte Hardware und Software Entwicklung mit Unterstützung von OpenCL, C und C++. Es wurde aufgrund der steigenden Nachfrage nach einem universellen HW-SW-Entwicklungs-Flow für FPGA-Beschleuniger auf Basis des Eclipse Design Environments (IDE) für Datenzentren entwickelt.

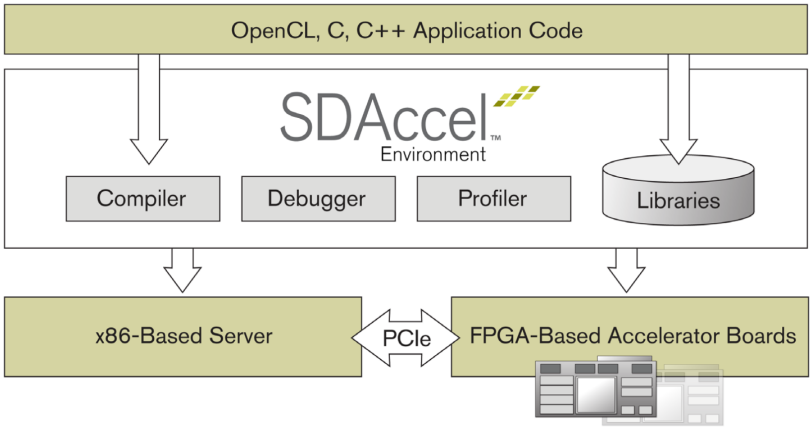


Abbildung 6.2: SDAccel Entwicklungsumgebung [xilb]

SDAccel umfasst einen HLS- und RTL-Compiler zur Optimierung von Streaming- und Datenpfad-Applikationen mit geringer Latenz bei effizientem Gebrauch der FPGA-Ressourcen. Der Compiler bildet die Berechnungs- und Datenstrukturen des OpenCL Models auf die konfigurierbaren LUT, DSP und Speichereinheiten des FPGAs ab (siehe Abbildung 6.3).

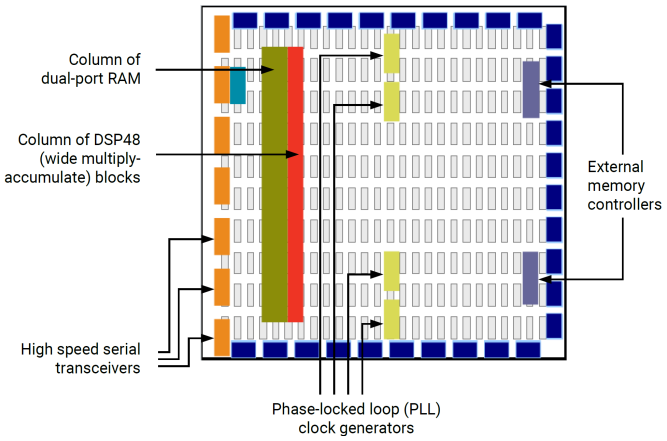


Abbildung 6.3: Xilinx FPGA Architektur [xilc]

Die integrierte SW-Emulation und HW-Emulation verringert die Entwicklungszeit von HW/SW-Anwendungen auf FPGA-Beschleunigern.
Durch die Unterstützung der dynamischen Rekonfiguration von Xilinx FPGA-Devices können Anwendungen (Kernel) auf einer FPGA-Beschleunigerkarte zur Laufzeit ausgeführt werden. Dazu wird ein statischer Teil für die initiale Konfiguration der PCIe Schnittstelle und der Onboard-Speicher verwendet. Zur Laufzeit kann dann über die PCIe Schnittstelle ein partielles Bitfile mit der Nutzer-Anwendung geladen werden. Ein integrierter Scheduler kann zur Laufzeit mehrere parallele Kernel betreiben und auf verfügbare Ressourcen einer FPGA-Karte aufteilen.
Die Kernel kommunizieren dabei über das interne AXI-Bussystem mit den Onboard-Speichern, dem Host-System und anderen Beschleuniger-Kerneln. Abbildung 6.4 zeigt den Aufbau mit statischer und dynamischer FPGA Region in der SDAccel Umgebung.

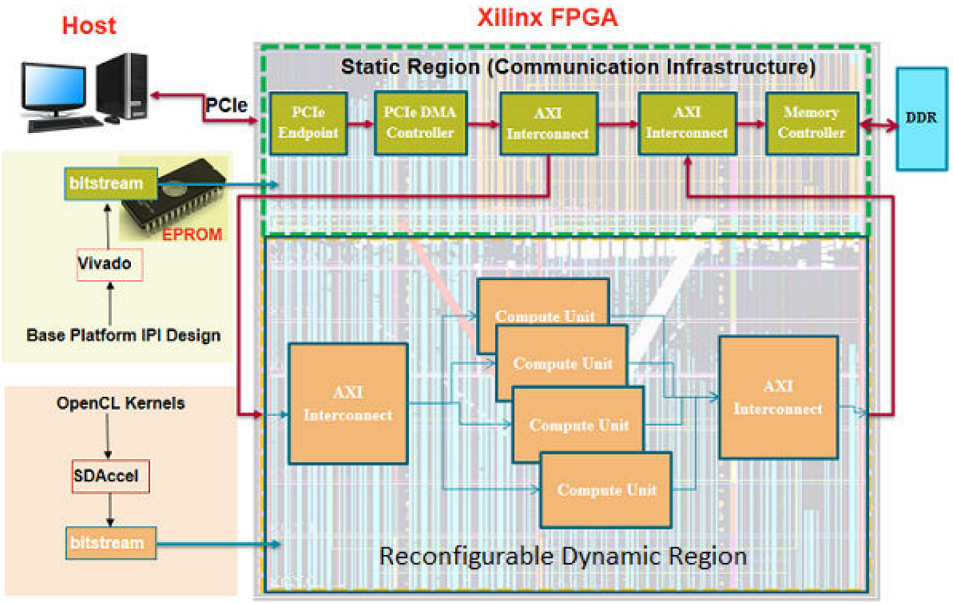


Abbildung 6.4: Xilinx Dynamic Region Beispiel [xilc]

Zur Optimierung des Datenflusses unterstützt SDAccel Dataflow und Pipelining Anweisungen. Damit können Lese- und Schreibzugriffe parallel zu den Berechnungsteilen ausgeführt werden, um einen bestmöglichen Datendurchsatz und eine konstante Auslastung der Module zu erreichen. In Abbildung 6.5 ist das Datenfluss Modell von SDAccel dargestellt.
Der Aufbau der internen Block-RAM-Speicher und die Speicherport-Konfiguration wird in SDAccel über Array Partitionen abgebildet und angepasst. Ein Speicher-Array kann dabei mit Block-Partitionierungen, zyklischen Partitionierung und vollständiger Partitionierung konfiguriert werden. Abbildung 6.6 zeigt die verschiedenen Speicher-Partitionierungen in SDAccel.

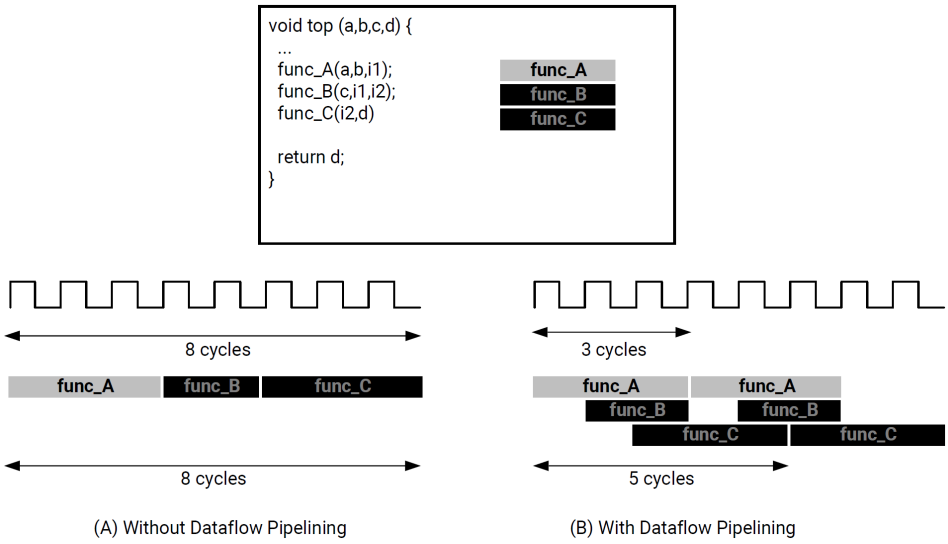


Abbildung 6.5: SDAccel Dataflow Beispiel [xilc]

Studien der ETH Zürich in Zusammenarbeit mit Xilinx [xilb] haben gezeigt, dass FPGA-basierte Beschleuniger eine bis 25x bessere Leistungseffizienz (Performance/Watt) und eine 50-75x geringere Latenz erreichen können als CPU/GPU Implementierungen.
Abbildung 6.7 zeigt den Performance-Vergleich von CPU, GPU und FPGA-Beschleunigern für DNN (Deep Neural Network) Vorhersage-Systeme in Web-Anwendungen aus einer Präsentation von Baidu [xilb] auf dem 2014 Hot Chips Symposium. Dabei erreichen FPGA-Beschleuniger eine bis zu 22 fach bessere Performance/W im Vergleich zu GPU-Systemen.

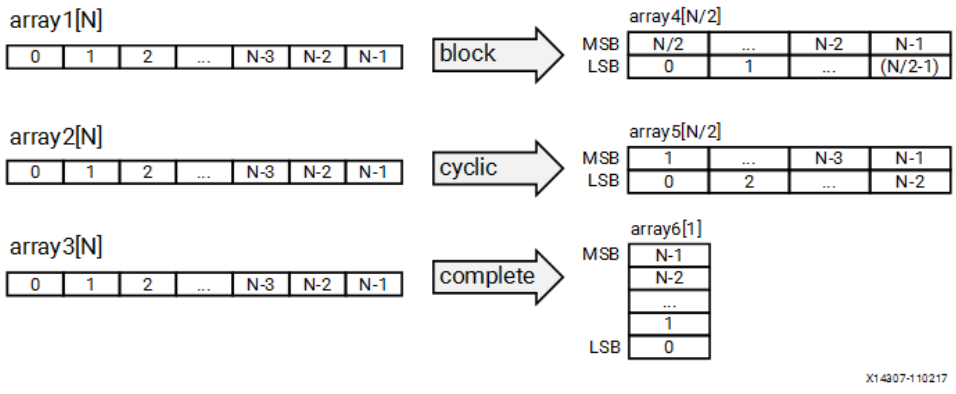


Abbildung 6.6: SDAccel Array Partitionierung [xilc]

Das SDAccel Framework bietet eine ideale Grundlage für die Entwicklung von Hardware-Beschleunigern mit hoher HW/SW Interaktion wie in der

SLAM-Anwendung dieser Dissertation gefordert. Merkmalsbasierte SLAM-Systeme stellen hohe Anforderungen an das Speichersystem und die Kommunikation der einzelnen Komponenten. Wenn eine Echtzeit-Anforderung besteht, steigen die Anforderungen zusätzlich.
Zur Deskriptor-Generierung werden meist durch die Verteilung der Merkmalspunkte über das Bild und durch die Rotation des Deskriptors ein Vielzahl an Speicheradressierungen parallel ausgeführt. Mit Unterstützung des OpenCL Speicher-Modells ist die Xilinx SDAccel HW/SW Co-Design Umgebung ein gutes Tool zur Optimierung und zum Entwurf eines Beschleuniger-Systems für merkmalsbasierte SLAM-Anwendung.

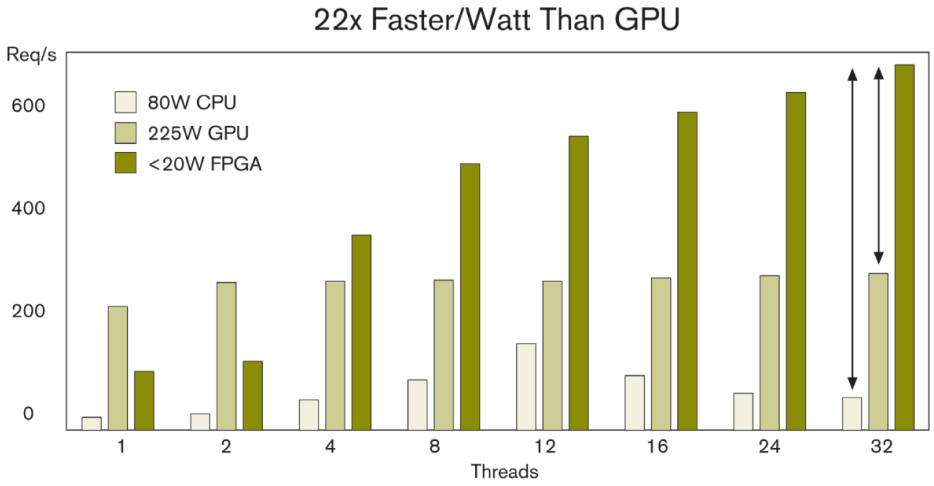


Abbildung 6.7: FPGA/GPU/CPU Performance/Watt in DNN Predictor System [xilb]

6.2.1 KCU1500 FPGA-Beschleuniger Karte

In dieser Arbeit wird das Kintex UltraScale FPGA Acceleration Development Kit KCU1500 als Zielplattform verwendet. Das Kit enthält ein PCIe Beschleuniger Board mit einem XCKU115-2FLVB2104E FPGA und 16GB DDR4 Speicher.
Die Kommunikation mit einem Host-Computer kann über zwei x8 PCIe Gen3 Schnittstellen realisiert werden. Darüber hinaus können durch die zwei QSFP28 Stecker seriellen High-Speed Verbindungen wie z.B. 40Gb Ethernet zu externen Geräten aufgebaut werden. Abbildung 6.8 zeigt das KCU1500 Board.
Das KCU1500 Board im Single Slot PCIe Format wird als Referenz-Plattform in der SDAccel Umgebung unterstützt. Damit kann es für System-Designs mit FPGA-Beschleunigern in einer Vielzahl von Anwendungen wie z.B. Video Verarbeitung, Datenanalyse, künstliche Intelligenz und maschinelles

Abbildung 6.8: Xilinx KCU1500 FPGA-Beschleuniger Karte [xila]

Lernen eingesetzt werden.
Durch die vergleichsweise geringe Stromaufnahme des FPGAs kann das KCU1500 auch in mobilen Systemen oder für Systeme mit geringen Energie-Kapazitäten eingesetzt werden.
Der Kintex KU115 FPGA des KCU1500 bietet eine große Zahl an verfügbaren Ressourcen. Tabelle 6.1 gibt eine Zusammenfassung der Eigenschaften des XCKU115-2FLVB2104E FPGAs.

Logikzellen	1,451 Mio
DSP Slices	5520
Block RAM (Mb)	75,9
16.3Gb/s Transceivers	64
I/O Pins	832

Tabelle 6.1: Eigenschaften des XCKU115-2FLVB2104E FPGAs

In dieser Arbeit wird das KCU1500 als Beschleuniger für die vorgestellte 6-DoF SLAM Anwendung eingesetzt. Durch die schnelle PCIe Schnittstelle können die Bilddaten mit hohen Datenraten übertragen werden und die Ergebnisse der Merkmalsextraktion direkt zurück gelesen werden. Der große Block-RAM Speicher kann sehr gut als Bildspeicher genutzt werden. Im SDAccel System bietet das KCU1500 Board durch die dynamische Rekonfiguration und die Programmierung des Xilinx FPGAs über die PCIe-Schnittstelle, eine GPU-nahe Entwicklungsmöglichkeit und die Basis für ein effizientes Beschleuniger-System.

6.3 GPU Implementierung

Als Referenzimplementierung zur einfacheren Portierung auf die FPGA-Beschleuniger Plattform wurde für diese Arbeit eine Architektur mit GPU-Beschleuniger gewählt. Dazu wurden die in Kapitel 5.1 vorgestellten Verfahren als OpenCL Kernel implementiert und mit einer Host-Software in das RTAB-Map SLAM System integriert. Abbildung 6.9 zeigt eine Übersicht der Merkmalsextraktion mit GPU-Beschleuniger.

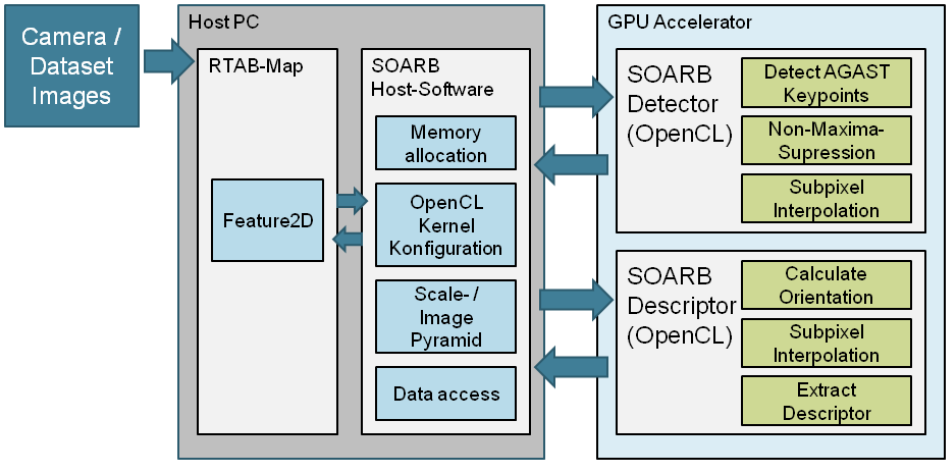


Abbildung 6.9: Übersicht der GPU-Implementierung

Die Host-Software übernimmt dabei die Konfiguration der OpenCL Kernel, die Speicherallokation und die Übergabe der Zeiger an die Beschleuniger-Komponenten. Die Host-Software erstellt ebenfalls die SOARB-Skalenpyramide in einem Vorverarbeitungsschritt.
Der SOARB-Detektor Kernel beinhaltet einen modifizierten AGAST Detektor mit der präsentierten Subpixel-genauen Interpolation der Merkmalsposition. Die Verarbeitung läuft parallel für jeden Bildpunkt ab und nutzt somit die maximalen verfügbaren Recheneinheiten der GPU aus. Die Interpolation wurde wie in Abschnitt 5.1.1 gezeigt in die Non-Maxima Unterdrückung integriert.
In dem SOARB-Deskriptor Modul wird für jedes Merkmal in der Übergabe-Liste ein Merkmalsdeskriptor erstellt. Unter Ausnutzung der parallelen Recheneinheiten wird hier für jedes Merkmal eine Orientierung und der rotierte Deskriptor mit Subpixel-genauer Berechnung der Vergleichsintensitäten erstellt.
Die Ergebnisse der GPU Referenzimplementierung und die Ausführungszeiten werden in den Kapiteln 8.1 und 8.2 betrachtet.

6.4 FPGA Implementierung

Auf Basis der OpenCL GPU Implementierung wurde eine angepasste Version des SOARB-Algorithmus für Xilinx SDAccel entworfen. Die Detektor und Deskriptor Komponenten des SOARB-Verfahrens wurden dazu als getrennte OpenCL-Kernel implementiert, die in Kombination mit einer C++ Host-Software geladen und ausgeführt werden.

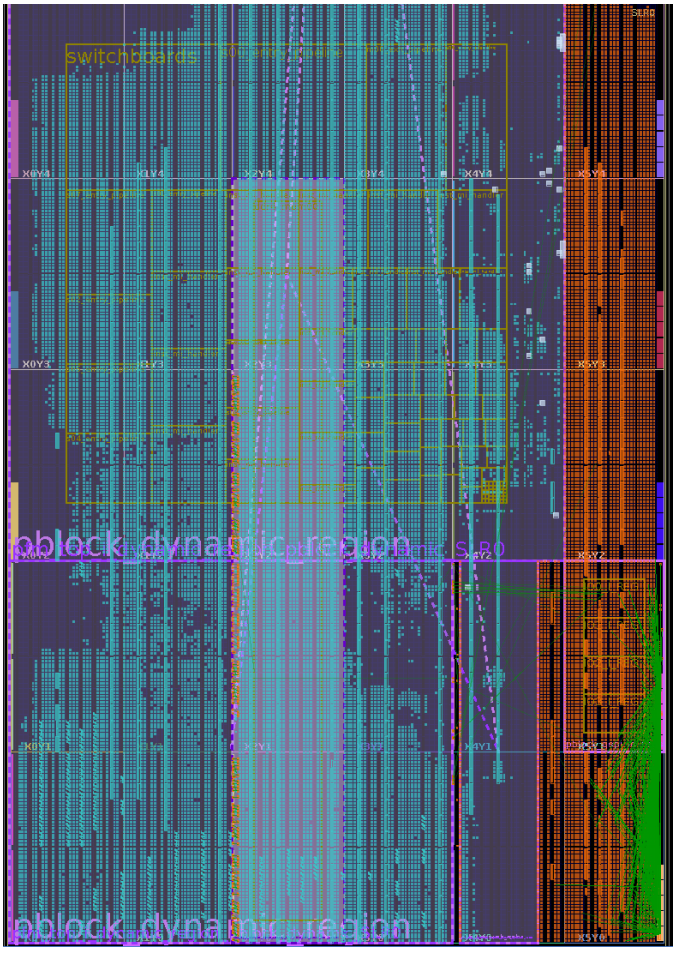


Abbildung 6.10: FPGA Layout der SOARB Detektor und Deskriptor Kernel

Die Host-Software führt dabei die Konfiguration der Kernel und die Erstellung der Skalenpyramide aus. Die Kommunikation mit den Hardware-Modulen erfolgt über den PCIe-Bus des Host-Computers mit Hilfe einer PCIe zu AXI Bridge. Die Kernel werden dabei erst zur Laufzeit in die rekonfigurierbare Region des FPGA-Beschleunigers geladen. Abbildung 6.10 zeigt das implementierte Design mit auf der Xilinx KCU1500 Beschleuniger-Karte mit XCKU115-2FLVB2104E FPGA.

6.4.1 SOARB-Detektor

Für den SOARB-Detektor wurde eine Pipeline-Verarbeitung entworfen, um einen hohen Datendurchsatz und stetige Auslastung der Hardware-Module zu gewährleisten. Dazu wurde eine Speicherarchitektur mit Zeilenspeicher für 7 Zeilen mit Registern und Block-RAM Speichern gewählt. Das 7x7 Pixel Fenster für die FAST/AGAST 16-Pixel Segment Tests wurde als Registerfenster mit Schieberegister realisiert, um gleichzeitig Zugriff auf alle relevanten Pixel zu haben, und damit alle Pixel-Vergleiche parallel berechnen zu können. Pixel am Ende des Register-Fensters werden in einen Block-RAM Zeilen-Speicher geladen. Am Ende der Zeile wird das Schieberegister der nächsten Zeile gefüllt. Der Ablauf der Detektion mit dem verwendeten Speicheraufbau ist in Abbildung 6.11 dargestellt.

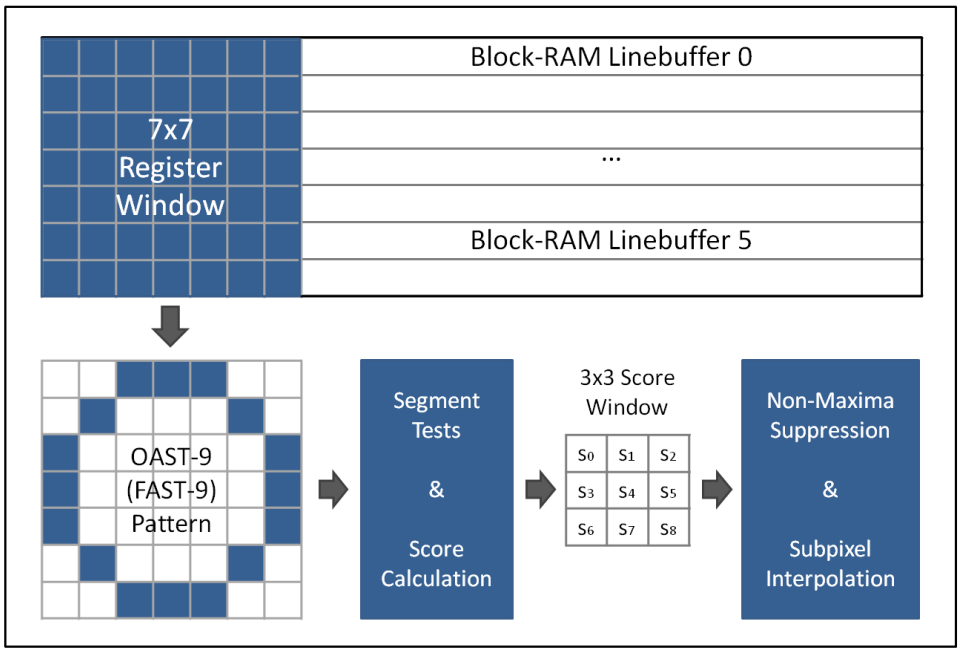


Abbildung 6.11: Speicheraufbau und Pipeline-Architektur des SOARB Detektors (ohne parallele Verarbeitung)

Um einen besseren Speicherdurchsatz des DDR3 Speichers zu erreichen, sind große Bit-Breiten bei den Speicherzugriffen nötig. Im SOARB-Detektor werden daher pro Takt 16 Pixel parallel geladen (128Bit) und somit die Verarbeitung von jeweils 16 AGAST-Detektoren gleichzeitig ausgeführt. Dazu wurde das Registerfenster von 7x7 Pixel auf 32x7 Pixel vergrößert und der Zeilenspeicher entsprechend verkleinert. Die Ergebnisse der Segment-Tests werden nach der Score-Berechnung in einem 3x3 Register-Fenster mit 2 Block-RAM Zeilenspeichern zwischengespeichert. Über dieses Fenster wird dann die Non-Maxima Suppression ausgeführt, es wird also nur der höchs-

te Score als Merkmal markiert. Zusätzlich wird für jedes erkannt Merkmal die Subpixel-genaue Merkmalsposition nach dem in Kapitel 5.1.1 beschriebenen Verfahren bestimmt. Dabei konnten viele der Multiplikationen durch Schiebeoperationen ersetzt werden, um die Berechnungen auf der FPGA-Architektur weiter zu beschleunigen.

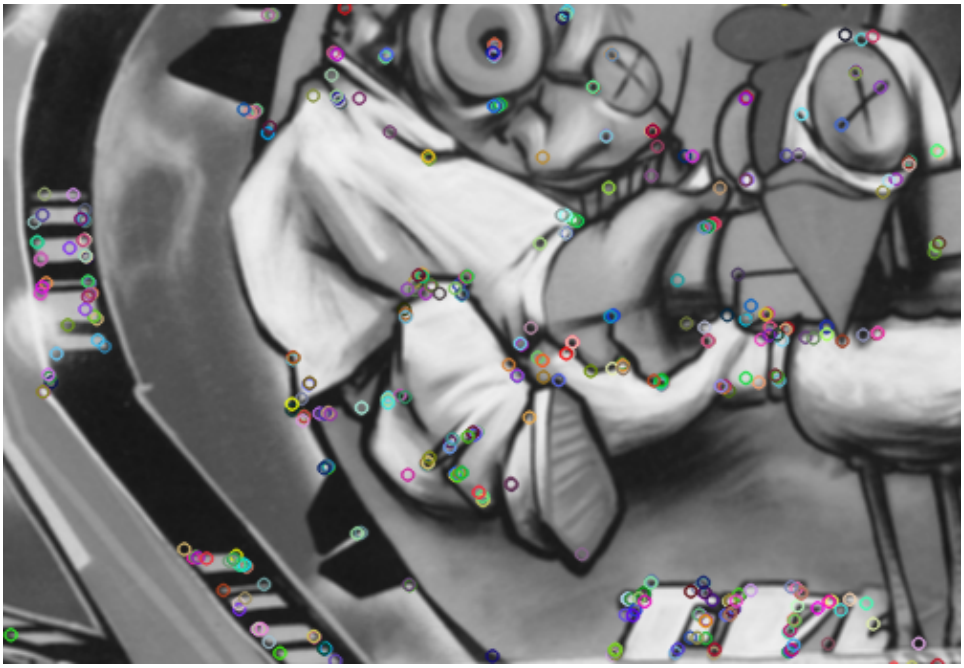


Abbildung 6.12: Ergebnis des SOARB-Detektors auf dem Grafitti-Bild des Oxford-Datasets

Damit der SOARB-Detektor Skalen-invariant ist, wird der AGAST-Detektor für jedes Bild der zuvor erstellten Skalenpyramide ausgeführt. Für jedes erkannte Merkmal werden dann die Merkmalsposition in Subpixel-Koordinaten, die Skalen-Ebene und der Score als Vektor-Liste zurück in den DDR3-Speicher geschrieben.

6.4.2 SOARB-Deskriptor

Der SOARB-Deskriptor wurde als getrenntes OpenCL Kernel-Modul implementiert, das von dem Host-Programm direkt im Anschluss an die Merkmalsdetektion ausgeführt wird. Die Berechnung wird für jedes Merkmal in der zuvor erstellten Merkmals-Liste ausgeführt. Dazu wird für jedes Merkmal ein 48x32 Pixel Fenster in einem Block-RAM Speicher aufgebaut, um die Speicher-Zugriffszeit für die nachfolgenden Pattern-Vergleiche zu verringern. Das Fenster wird dabei ebenfalls mit 16-Pixel Blöcken (128Bit Words) gefüllt, um zusätzlich die Anzahl der DDR3-Speicherzugriffe zu reduzieren.

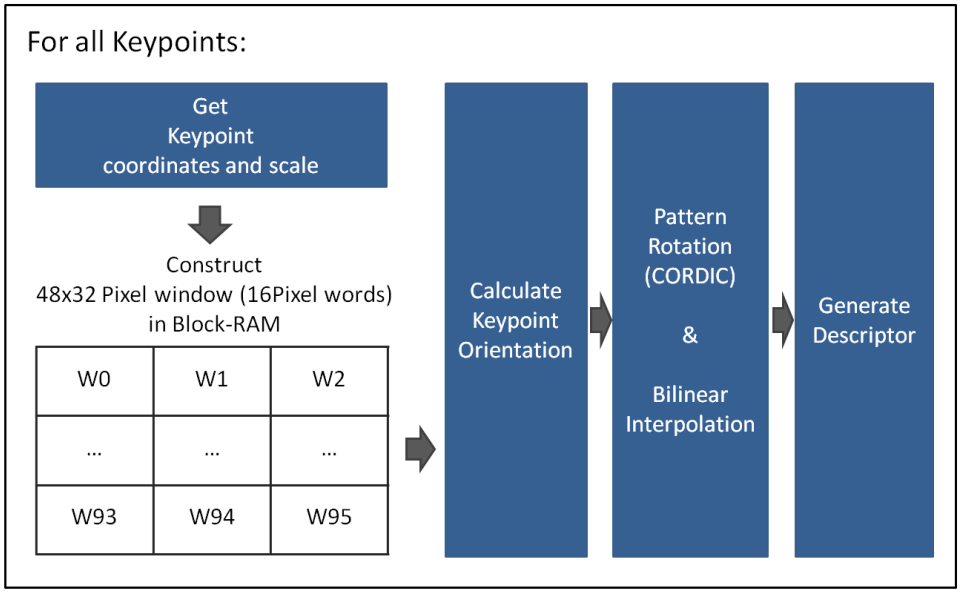


Abbildung 6.13: Ablauf des SOARB Deskriptors

Sobald das Fenster gefüllt ist, wird aus dem 31x31 Pixel Bereich um das Merkmal zunächst die Orientierung des Merkmals über den Intensity Centroid bestimmt. Die Bestimmung der Mermals-Orientierung wurde auf dem FPGA mit einem CORDIC Verfahren zur Tangens und Sinus/Cosinus Berechnung umgesetzt. Eine detailliertere Übersicht über das verwendeten Verfahren ist in Kapitel 6.4.3 aufgelistet. Mit dem Drehwinkel kann das rotierte ORB-Pattern in dem 31x31 Pixel Fenster über bilineare Interpolation ausgelesen und der Deskriptor über die 256 verteilten Intensitätsvergleiche erstellt werden. Der Ablauf der SOARB-Deskriptor Erstellung ist in Abbildung 6.13 gezeigt.

6.4.3 CORDIC Berechnungen zur Merkmals-Rotation

Das CORDIC-Verfahren ist ein Konvergenzverfahren auf Basis von Koordinatentransformation, das von Volder [Vol59] und Walther [Wal71] entwickelt wurde. Das Verfahren benötigt nur einfache Operationen wie Addition, Schiebeoperationen und Vergleiche, die sich sehr gut in Hardware-Schaltungen wie z.B. in FPGAs und ASICs umsetzten lassen.
Mit einer Koordinatentransformation lassen sich trigonometrische Funktionen durch Drehung eines Vektors $(x0; y0)$ um einen Winkel Θ in einen Vektor $(xn; yn)$ nach

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{6.1}$$

umformen. Mathematisch ist diese Funktion durch Multiplikation mit einer Rotationsmatrix durchführbar. Abbildung 6.14 zeigt die Rotation des Vektors.

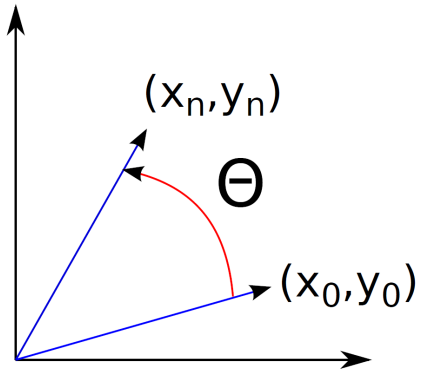


Abbildung 6.14: CORDIC Verfahren [uMS12]

Durch Drehung des Vektors $(1; 0)$ um den Winkel Θ lassen sich damit $\sin\Theta$ und $\cos\Theta$ berechnen. Zur Implementierung in Hardware-Schaltungen lässt sich die Drehung um den Winkel Θ als Linearkombination von vorab definierten Teilwinkeln realisieren. Dadurch wird die Drehung iterativ bis zu gewünschten Genauigkeit durch alternierende Approximation durch Addition oder Subtraktion von Teilwinkeln erreicht. Eine auf den gewünschten Drehwinkel initialisierte Hilfsvariable z_n wird eingesetzt, um die Drehrichtung bzw. das Vorzeichen der Addition iterativ über Vergleich mit dem Zielwert zu steuern.
Die Multiplikationen können in Hardware durch Schiebe-Operationen ersetzt werden. Dabei werden meist Fixed-Point Implementierungen gewählt, um die einzelnen Operationen mit geringem Ressourcenverbrauch und mit einer geringen Latenz umzusetzen. Die Addition der Teilwinkel ist in Abbildung 6.15 dargestellt.

Man unterscheidet bei der CORDIC-Implementierung zwischen Rotationsmodus und Vektormodus. Bei dem Vektormodus wird statt der z-Komponente die y-Variable gegen null konvergiert. Nach Aufsummieren der Teilwinkel enthält z_n den Winkel identisch mit $arctan(y0 = x0)$. Damit lassen sich Funktionen wie $atan2$, $arcsin$ und $arccos$ berechnen.

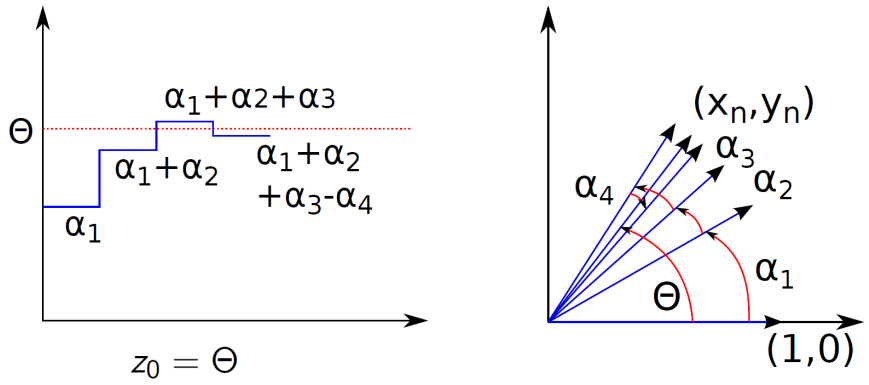


Abbildung 6.15: Addition der Teilwinkel [uMS12]

CORDIC-Verfahren bieten eine sehr gute Möglichkeit zur Berechnung von trigonometrischen Funktionen bei konfigurierbarer Latenz und Genauigkeit über die Anzahl der Iterationen. Bei einer guten Auswahl der Initialwerte lassen sich somit gute Ergebnisse für Merkmals-Verfahren erreichen. Als Alternative werden zur Berechnung von trigonometrischen Funktionen häufig auch Look-Up Tabellen verwendet. Diese benötigen allerdings meist eine aufwändigere lineare Interpolation, um eine vergleichbare Genauigkeit zu erreichen. Daher wurde für die Berechnung der ATAN2-Funktion zur Bestimmung der Merkmals-Rotation und für die Sinus- und Cosinus-Berechnung zur Rotation der Deskriptor-Testpunkte jeweils ein CORDIC-Verfahren eingesetzt.

Bestimmung der Merkmals-Orientierung

Die Orientierung des Merkmals wird in dem SOARB-Algorithmus über einen Intensity Centroid aus den Momenten m_{01} und m_{10} in X-Richtung und Y-Richtung des 31x31 Pixel Bildausschnitts der Merkmalsumgebung berechnet:

$$\theta = atan2(m_{01}, m_{10}) \tag{6.2}$$

Für die Implementierung auf der FPGA-Beschleunigerkarte wurde die Berechnungen der Momente und des Winkels auf die FPGA-Architektur optimiert. Dazu wurden Festkomma-Berechnungen und breitere Datentypen für die Übertragung des Bildausschnittes verwendet.

```
short calc_orientation(uint8 *patch){
             int m_01 = 0, m_10 = 0;
            __attribute__((xcl_pipeline_loop))
            for(int v = 0; v <= 15; v++ )
            {
                int d = UMAX(v);
                uchar pdata0[32]__attribute__((xcl_array_partition(
                    complete,0)));
                uchar pdata1[32]__attribute__((xcl_array_partition(
                    complete,0)));
                uint8_to_uchar(patch[15+v],pdata0);
                uint8_to_uchar(patch[15-v],pdata1);
                for(int u = 0; u <= d; u++ )
                {
                    int val_pp= pdata0[15+u], val_mp = pdata1[15+u];
                    int val_pm= pdata0[15-u], val_mm = pdata1[15-u];
                    m_01 += ((val_pp-val_mp)+(val_pm-val_mm))*v;
                    m_10 += ((val_pp-val_pm)+(val_mp-val_mm))*u;
                }
            }
            short angle = cordic_atan2(fixed18_9(m_01), fixed18_9(m_10
                ));
            //if(m_01<0) angle = 65536 + angle;
            return angle;
}
```

Listing 6.1: SOARB Orientierung(OpenCL/SDAccel)

Listing 6.1 zeigt den Ablauf der Momentberechnung über gewichtete Summierung und der Winkelberechnung mit einer CORDIC-Implementierung der ATAN2 Funktion. UMAX ist ein Array, das die Zeilenlängen für die Kreisfläche des Zentroiden enthält.
Die CORDIC Implementierung der Atan2-Funktion ist eine angepasste Portierung zu OpenCL aus der Xilinx Hlx Bibliothek. Sie verwendet Festkomma-Berechnungen mit 18Bit und 9 Bit Integer-Anteil für die Koordinaten und 16Bit mit 4Bit Integer-Anteil für die Phasen. Sie bieten vollständiges Pipelining mit einer Latenz von 12 Takten (Pro Takt ein Ergebnis) und einen mittleren Fehler von 0,106 Grad über die mitgelieferten Testdaten. Damit ist die CORDIC-Implementierung genauer als die in ORB verwendete OpenCV Funktion $fastAtan2$ mit einer Genauigkeit von ca. 0,3 Grad. Für die SOARB Implementierung wurden die Quadranten der HLS Implementierung angepasst, um Kompatibilität zu der fastAtan2 Funktion zu bieten. Listing 6.2 zeigt die in SOARB verwendete CORDIC Atan2 Implementierung.

```
short cordic_atan2(int y0, int x0)
{
    int x, y, xp, yp;
    short z=0, zn=0, zp;
    bool dneg;

    const short cord_M_2PI  = fixed16_4(6.28318530717958647693);
    const short cord_M_PI   = fixed16_4(3.14159265358979323846);
```

```
    const short cord_M_PI_2 = fixed16_4(1.57079632679489661923);
    const short atan_2Mi[] = { fixed16_4(0.7854), fixed16_4(0.4636),
        fixed16_4(0.2450), fixed16_4(0.1244), fixed16_4(0.0624),
        fixed16_4(0.0312), fixed16_4(0.0156), fixed16_4(0.0078),
        fixed16_4(0.0039) };

    if (x0 < 0) x = -x0;
    else        x =  x0;

    if (y0 < 0) y = -y0;
    else        y =  y0;

    if (y0 == 0) // SPECIAL CASE Y==0
    {
        if (x0 < 0)  zn = cord_M_PI;
        else         zn = 0;
        return zn;
    }

    if (x0 == 0) // SPECIAL CASE X==0
    {
        if (y0 < 0) zn = -cord_M_PI_2;
        else        zn =  cord_M_PI_2;
        return zn;
    }
    __attribute__((xcl_pipeline_loop(1)))
    for (int i = 0; i <= ROT; i++)
    {
        dneg = (y > 0);
        if (dneg) {
            xp = x + (y >> i);
            yp = y - (x >> i);
            zp = z + atan_2Mi[i];
        } else {
            xp = x - (y >> i);
            yp = y + (x >> i);
            zp = z - atan_2Mi[i];
        }
        x = xp;
        y = yp;
        z = zp;
    }
    if ((x0 > 0) & (y0 > 0)) {          //QUADRANT 1
        zn = z;
    } else if ((x0 < 0) & (y0 > 0)) {  //QUADRANT 2
        zn = cord_M_PI - z;
    } else if ((x0 < 0) & (y0 < 0)) {  //QUADRANT 3
        zn = cord_M_PI + z;
    } else if((x0 > 0) & (y0 < 0)) {   //QUADRANT 4
        zn = cord_M_2PI - z;
    }
    return zn;

}
```

Listing 6.2: SOARB CORDIC Atan2(OpenCL/SDAccel)

Rotation des Deskriptor-Patterns

Das SOARB Deskriptor-Pattern wird zur Rotations-invarianten Konstruktion des Deskriptors um die Merkmals-Orientierung gedreht. Für eine Merkmals-Orientierung θ und die Pattern-Koordinaten p_x, p_y werden die rotierten Pattern Koordinaten $p_{x'}, p_{y'}$ nach

$$\begin{aligned} p_{x'} &= p_x * cos(\theta) - p_y * sin(\theta) \\ p_{y'} &= p_x * sin(\theta) + p_y * cos(\theta) \end{aligned} \tag{6.3}$$

berechnet. Dazu sind eine Sinus und eine Cosinus Berechnung erforderlich. Zur Implementierung auf dem FPGA-Beschleuniger wurde daher eine CORDIC Implementierung der kombinierten *sincos* Funktion entworfen. In Listing 6.3 ist die in dem SOARB Deskriptor Kernel verwendete CORDIC SinCos Implementierung gezeigt.

```
short cordic_sincos(short angle, short *cos)
{
    int x, xp, y, yp;
    short z=0, zp;
    short xn =0, yn =0;
    bool dneg;

    const short cord_M_2PI  = fixed16_4(6.28318530717958647693);
    const short cord_M_PI   = fixed16_4(3.14159265358979323846);
    const short cord_M_PI_2 = fixed16_4(1.57079632679489661923);

    const short atan_2Mi[] = {fixed16_4(0.7854), fixed16_4(0.4636),
        fixed16_4(0.2450), fixed16_4(0.1244), fixed16_4(0.0624),
        fixed16_4(0.0312), fixed16_4(0.0156), fixed16_4(0.0078),
        fixed16_4(0.0039)};

    x = fixed16_4(0.60726);
    y = 0;

    if (angle > cord_M_PI_2)
    {
        z = cord_M_PI;
    }
    else if (angle > (cord_M_PI+cord_M_PI_2))
    {
        z = cord_M_2PI;
    }

    __attribute__((xcl_pipeline_loop(1)))
    for (int i = 0; i < ROT; i++)
    {
        if (angle > z) {
            xp = x - (y >> i);
            yp = y + (x >> i);
            zp = z + atan_2Mi[i];
        } else {
            xp = x + (y >> i);
            yp = y - (x >> i);
            zp = z - atan_2Mi[i];
```

```
        }
        x = xp;
        y = yp;
        z = zp;
    }

    if ((angle > cord_M_PI_2) & (angle < (cord_M_PI+cord_M_PI_2)))
    {
        xn = -x;
        yn = -y;
    }
    else{
        xn = x;
        yn = y;
    }

    *cos = xn;
    return yn;
}
```

Listing 6.3: SOARB CORDIC SinCos(OpenCL/SDAccel)

6.4.4 Optimierung von OpenCL-Kerneln

Neben den hier beschriebenen Optimierungsmöglichkeiten für die Umsetzung der OpenCL-Kernel in Hardware wurden weitere Optimierungen nach dem Xilinx SDaccel SDAccel Environment Profiling and Optimization Guide [xilc] durchgeführt. So benötigt der HLS-RTL-Compiler eine ein-dimensionale NDRange bzw. eine Work-Group Größe von (1,1,1), um die bestmögliche Optimierung und Parallelität zu erreichen. Darum wurden alle globalen und lokalen Arbeitsbereiche, soweit möglich, als Schleifen innerhalb des Kernels implementiert.

Kapitel 7

Hardware-Beschleunigung von SLAM-Systemen

In diesem Kapitel sollen Möglichkeiten zur Integration von Hardware-Beschleunigern in SLAM-Systeme untersucht werden. Kapitel 7.1 beschäftigt sich mit der Architektur von Merkmals-basierten SLAM-Systemen und zeigt Ansatzpunkte zur Integration einer Hardware-Beschleunigung sowie die Herausforderungen.
In Kapitel 7.2 wird die Integration eines FPGA-Beschleunigers mit der vorgestellten SOARB-Merkmalserkennung in das RTAB-Map SLAM System beschrieben.

7.1 Integration von Hardware-Beschleunigern

Merkmalsbasierte 6DoF SLAM-Verfahren wie RTAB-Map [LM11] und ORB-SLAM2 [MAT17] nutzen die Merkmalserkennung zur Lokalisierung und Kartographierung anhand von Bildinformationen. Dabei werden die Bildmerkmale, wie in Kapitel 2.1 gezeigt, in unterschiedlichen Komponenten des SLAM-Systems mit unterschiedlichen Anforderungen verwendet. So benötigt die visuelle Odometrie Bildmerkmale mit einer hohen Bildrate bzw. Update-Rate, um zuverlässig die Kameraposition zu verfolgen. Die Kartenerstellung und das Loop-Closure hat geringere Anforderungen an die Bildrate, aber fordert neben einer guten Qualität und Wiederholbarkeit der Merkmale auch zusätzlich eine verbesserte Rotations- und Skaleninvarianz.
Somit könnten nach den Anforderungen auch zwei getrennte Merkmalserkennungs-Verfahren für den jeweiligen Anwendungsbereich genutzt werden. Um die Berechnungszeit für die Erstellung der Merkmale zu reduzieren, wird jedoch meist versucht, die Merkmale der visuellen Odometrie für die darauffolgenden Teile der SLAM-Implementierung wiederzuverwenden. Daher wird ein Merkmalserkennungs-Verfahren mit hohen Bildraten und gleichzeitig guter Qualität benötigt.

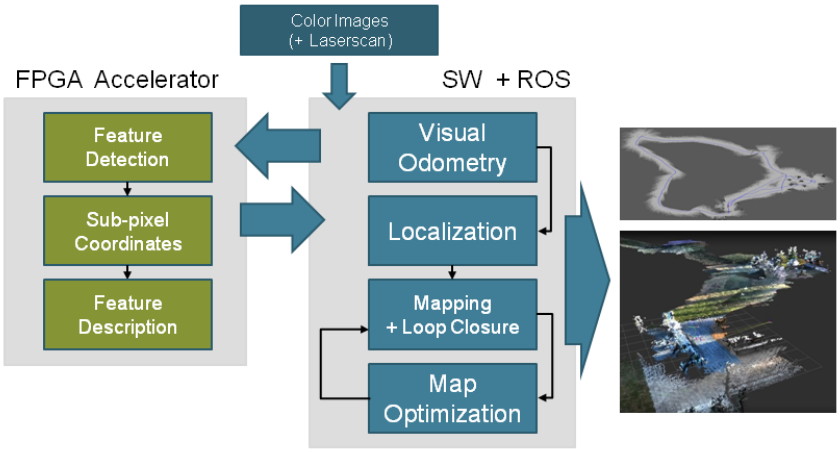


Abbildung 7.1: Integration von Beschleunigern in ein SLAM-System

Hardware-Beschleuniger wie z.B. FPGA-Beschleuniger und GPUs sind prädestiniert für diesen Anwendungsfall, da hier durch eine Parallelverarbeitung sehr hohe Wiederholungsraten trotz gleichbleibender oder höherer Komplexität erreicht werden können.
Abbildung 7.1 zeigt die Bestandteile eines SLAM-Systems mit integriertem Hardware-Beschleuniger.
In einem SLAM-System sollte die Merkmalsextraktion möglichst nah an die Merkmalsverarbeitung integriert werden, um Latenzen zu vermeiden. Daher sollte ein Beschleuniger direkt in das System eingebunden werden, um die Datenübertragung zwischen Host und Beschleuniger mit hohen Übertragungsraten zu erlauben.
Hier ist die OpenCL Architektur vorteilhaft, da durch HW-SW-Codesign eine bestmögliche Integration und Optimierung der Abläufe möglich wird.

7.2 RTAB-Map mit FPGA-basierter Merkmalserkennung

Das KCU1500 FPGA Board (siehe Bild 6.8) ist eine PCIe Erweiterungskarte, die direkt in das Host-System eingebettet werden kann. Über das PCIe Interface werden zur Laufzeit die vorkompilierten Hardware-Kernel sowie wichtige Parameter wie Speicheradressen und Kernel-Konfiguration geladen. Aufgrund der Unterstützung des OpenCL Frameworks ist die Integration der Host-Komponenten sehr ähnlich zu einer Konfiguration mit GPU-Beschleuniger. Die Komplexität der Anbindung eines herkömmlichen FPGA-Hardware Designs kann dadurch weitgehend vermieden werden.
Für diese Arbeit wurde ein RTAB-System mit FPGA-Beschleuniger ent-

worfen. Das KCU1500 Board enthält die SOARB Detektor- und SOARB-Deskriptor-Kernel als Hardware-Module. Zusätzlich können in den FPGA Teil auch noch weitere Beschleuniger eingebettet werden. So kann beispielsweise auch eine Stereo-Bildverarbeitung wie in Kapitel 3.5.1 beschrieben integriert werden, um die Verarbeitung der Stereo-Eingangsbilder zu beschleunigen.

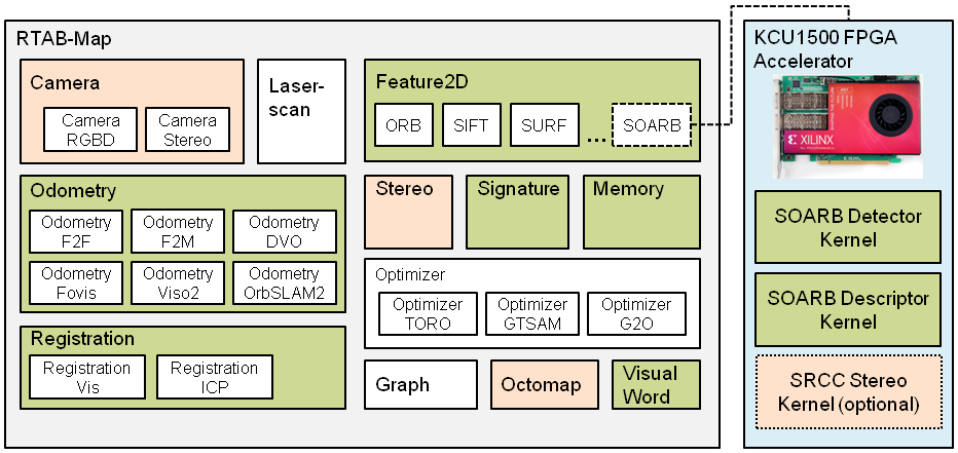


Abbildung 7.2: Integration des FPGA-Beschleunigers in RTAB-Map

Abbildung 7.2 zeigt das RTAB-Map System mit integriertem FPGA-Beschleuniger. Alle grün hinterlegten Komponenten benutzen die Merkmalserkennung bzw. profitieren von der Beschleunigung. Alle orange gefärbten Komponenten würden von einer Stereo-Verarbeitung in Hardware Nutzen erzielen.
In dieser Arbeit soll jedoch als Schwerpunkt die Integration einer beschleunigten Merkmalsextraktion betrachtet werden. Dazu wurden die Software-Komponenten zur Kommunikation mit den Beschleuniger-Kerneln und zur Konfiguration der FPGA-Karte in das Modul Feature2D integriert.
Dieses Modul wird von allen RTAB-Map Komponenten zur Merkmalsverarbeitung verwendet und enthält auch die Schnittstellen zu anderen Merkmals-Detektoren und Deskriptoren. Somit bietet sich hier eine direkte Einbindung des Beschleunigers mit allen erforderlicher Daten zur Merkmalsverarbeitung an.

Kapitel 8

Ergebnisse

8.1 Evaluation des SOARB Detektors/ Deskriptors

Die GPU Implementierung des vorgestellten SOARB-Algorithmus wurde mit dem Oxford-Dataset für affine kovariante Regionen [MTS+05] evaluiert und gegen andere in OpenCV enthaltene GPU-basierte Verfahren der Bildmerkmals-Detektion und -Deskription verglichen. Das Dataset enthält Homographien der Transformationen von jeweils 2 Bildpaaren als Ground Truth. Daraus können Abweichungen und das Inlier-Verhältnis für jeden Algorithmus bestimmt werden. Das Dataset besteht aus Farbbildern, daher konnte ebenfalls der SOARB-RGB Algorithmus mit Farbinformationen untersucht werden. Alle herkömmlichen Algorithmen reduzieren die Eingangsbilder zu Graustufenbilder, da hier lediglich die Pixel-Intensitäten ohne Farbinformationen genutzt werden. Für die Evaluation wurden die ersten drei Bildpaare (H1-H3) jedes Datensatzes genutzt.

Detektor/ Deskriptor	∅ Merkmale	Inlier Ratio	∅Det. Zeit(ms)	∅Deskr. Zeit(ms)	Total (ms)	FPS
SOARB-RGB	1428	99.2%	9.5	11.2	20.7	48.3
SOARB	2264	97.9%	9.3	6.6	15.9	62.9
AKAZE	903	98.3%	34.0	31.1	65.1	15.4
ORB	1286	97.2%	6.3	7.2	13.5	74.0
SIFT	1358	96.9%	65.3	65.6	130.9	7.7
SURF	1131	95.2%	29.9	85.2	115.1	8.7

Tabelle 8.1: Evaluation der GPU-Implementierungen, Oxford Datasets H1-H3, Intel I7-6700K(3,6GHz) + GTX1050Ti

Tabelle 8.1 zeigt die durchschnittliche Anzahl der Merkmale und das Inlier Verhältnis mit einer maximalen Positionsabweichung von 4 Pixeln und einem

Nächsten-Nachbar Abstand von 0,6. Zusätzlich wurden die durchschnittlichen Laufzeiten für die Detektion und Deskription auf einem Intel Core i7 System mit 3,6GHz und einer dedizierten Nvidia GTX1050Ti Grafikarte mit OpenCL 2.0 Unterstützung bestimmt.
Die Ergebnisse zeigen eine sehr gute Erkennungsrate und Inlier Verhältnis des SOARB Algorithmus. Dabei konnte die Qualität der Ergebnisse trotz einer hohen Bildrate von über 60 Bildern/s gegenüber dem ORB Algorithmus weiter verbessert werden.
Im Vergleich zu den anderen getesteten Algorithmen mit GPU Unterstützung ist die Bildrate um Faktor 8x (SIFT), 7x (SURF) und 4x (AKAZE) höher. Es konnte dabei ein sehr gutes Inlier Verhältnis im Vergleich zu den komplexeren Algorithmen erzielt werden. Der SOARB-RGB Algorithmus erreicht das beste Inlier Verhältnis (99.2%) aller getesteten Implementierungen bei einer Bildrate von 48.3 Bildern/s.

8.1.1 Evaluation nach Mikolajczyk

Mit dem Oxford-Dataset wurde zusätzlich eine Evaluation des SOARB-Detektors und des SOARB-Deskriptors nach Mikolajczyk [MTS+05] [MS05] durchgeführt.
Für einen Merkmals-Detektor sind die Wiederholbarkeit und die Anzahl der gefundenen Merkmals-Korrespondenzenden gute Kriterien zur Bewertung der Merkmalserkennung.
Die Wiederholbarkeit ist definiert als der prozentuale Anteil der detektierten Merkmale, die eine korrekte Korrespondenz bzw. eine korrespondierende Region im anderen Bild haben. Die Anzahl der Region zu Region Korrespondenzen kann dabei aus der bekannten Homographie zwischen zwei Datensatz Bildern bestimmt werden.
Zur Bewertung von Merkmals-Deskriptoren definiert Mikolajczyk in [MS05] das Recall-Kriterium als die Anzahl der richtigen Matches gegenüber der Anzahl der korrespondierenden Regionen von zwei Bildern nach:

$$Recall = \frac{Number\ of\ correct\ matches}{Number\ of\ correspondences} \tag{8.1}$$

Die Anzahl der falschen Matches gegenüber der Anzahl der gesamten Matches definiert Mikolajczyk als 1-Precision:

$$1 - precision = \frac{Number\ of\ false\ matches}{Number\ of\ correct\ matches + Number\ of\ false\ matches} \tag{8.2}$$

Aus den Recall und 1-Precision Werten kann über eine Sammlung von Bildaufnahmen eines Datensatzes eine Recall/1-Precision Kurve erstellt werden. Mikolajczyk stellt für Matlab eine Evaluations-Umgebung bereit, mit der die in Matlab integrierten Merkmalserkennungen bewertet werden können. Die in dem Framework genutzten Funktionen wurden ebenfalls in der OpenCV Bibliothek implementiert.
Für diese Arbeit wurde mit OpenCV eine Detektor- und Deskriptor-Evaluation durchgeführt. Dazu wurde ein Framework erstellt, das die Detektoren und Deskriptoren von SOARB, ORB, SURF und SIFT mit dem Oxford Dataset miteinander vergleichen kann.

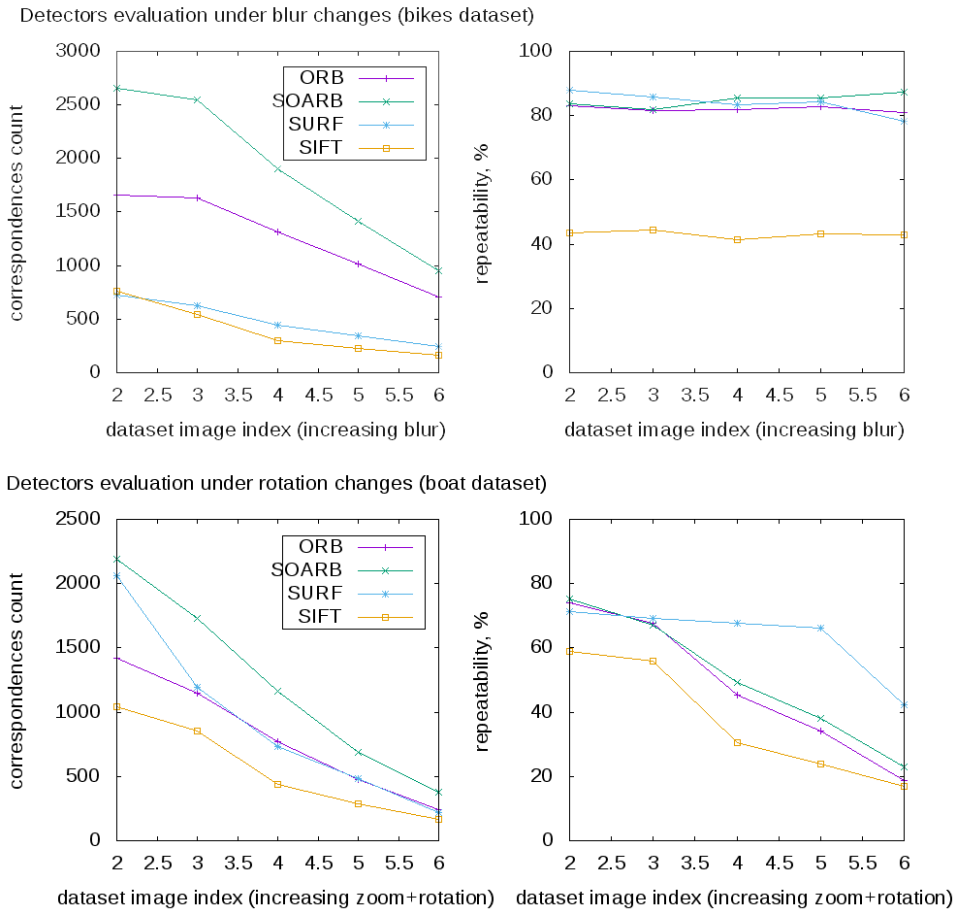


Abbildung 8.1: Oxford Detektor Evaluation - Anzahl der Korrespondenzen und Wiederholbarkeit - Bikes (Blur) und Boat(Zoom + Rotation)

Die Abbildungen 8.1 und 8.2 zeigen die Anzahl der Korrespondenzen und die Wiederholbarkeit der Detektoren für die Datensätze Bikes mit steigender Unschärfe (Blur), Boat mit steigender Rotation und steigendem Zoom-

Faktor, Grafitti mit Blickwinkel- und dem Leuven-Datensatz mit Beleuchtungsveränderungen.
Die Ergebnisse der Detektor-Evaluation zeigen, dass der SOARB-Detektor generell eine höhere Anzahl an Korrespondenzen über den Verlauf der Datensatz Bilder im Vergleich zu ORB bietet. Im Datensatz Leuven mit Varianz der Beleuchtung erreicht der SOARB-Detektor eine sehr gute Wiederholbarkeit im Verglich zu ORB. Hier erreicht der SOARB Detektor sogar die sehr guten Ergebnisse des SURF-Verfahrens.

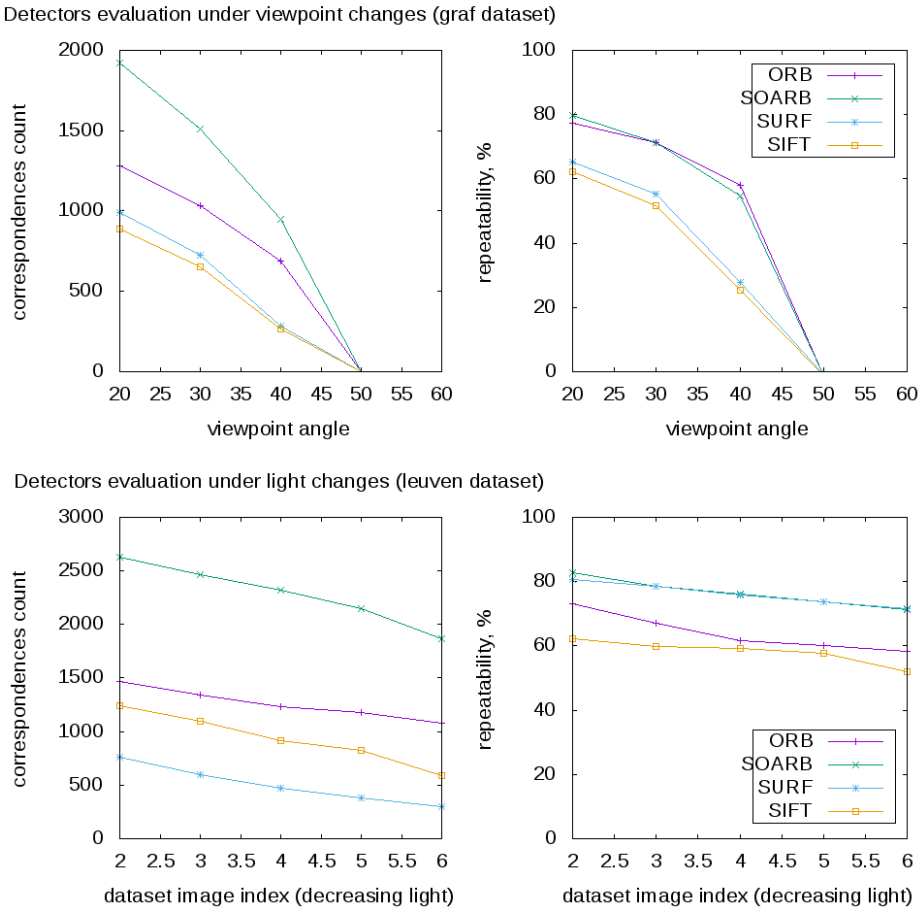


Abbildung 8.2: Oxford Detektor Evaluation - Anzahl der Korrespondenzen und Wiederholbarkeit - Grafitti (Viewpoint) und Leuven (Light)

Zur Deskriptor-Evaluation wurden mit dem OpenCV Framework die Recall/1-Precision Kurven für die verschiedenen Algorithmen zur Deskriptor-Generierung erstellt. Abbildung 8.3 zeigt die Recall/ 1-Precision Kurven für die Datensätze Bikes und Boat mit steigender Unschärfe (blur) sowie steigender Rotation und Zoom.

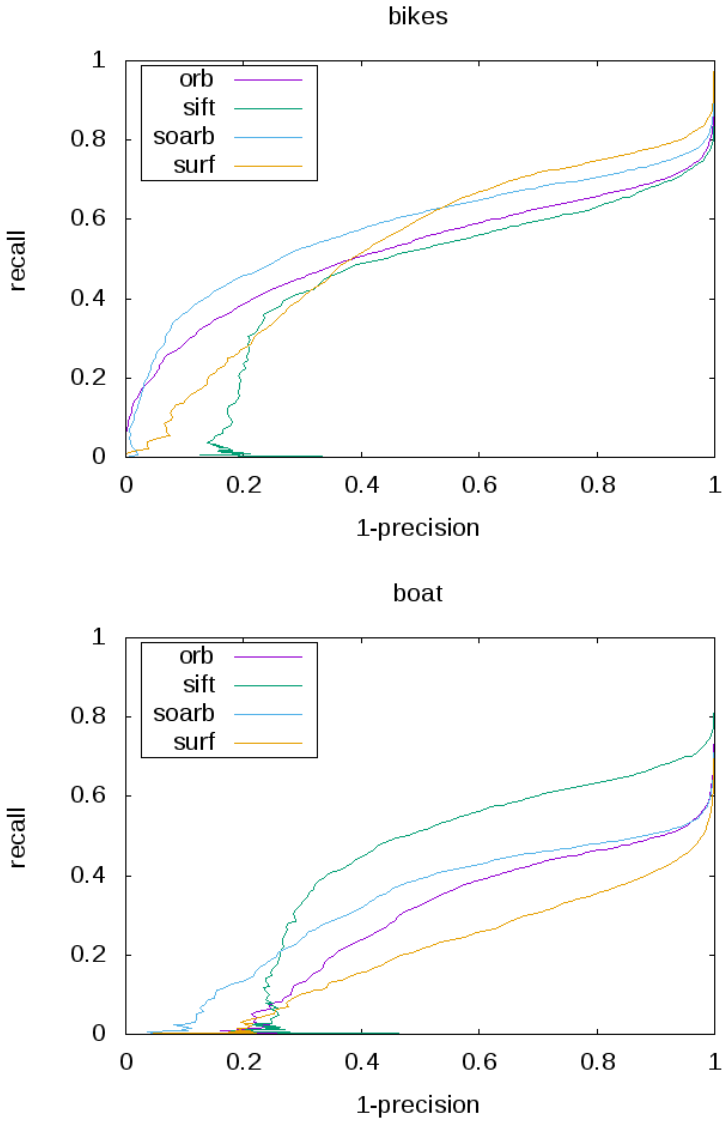


Abbildung 8.3: Oxford Deskriptor Evaluation - Recall- 1-Precision Kurven - Bikes (Blur) und Boat(Zoom + Rotation)

In Abbildung 8.4 sind die Ergebnisse für die verschiedene Blickwinkel (Grafitti) und steigender Beleuchtung (Leuven) dargestellt.

Die Ergebnisse zeigen, dass der SOARB Deskriptor in den Datensätzen sehr gute Invarianz zu den getesteten Bildveränderungen bietet. In jedem der Tests konnte eine Verbesserung im Vergleich zu dem ORB-Deskriptor festgestellt werden. Hier zeigt sich, dass die Subpixel-Genauigkeit bei der Deskriptor Generierung deutliche Vorteile im Vergleich zu den anderen getesteten

Verfahren bietet.

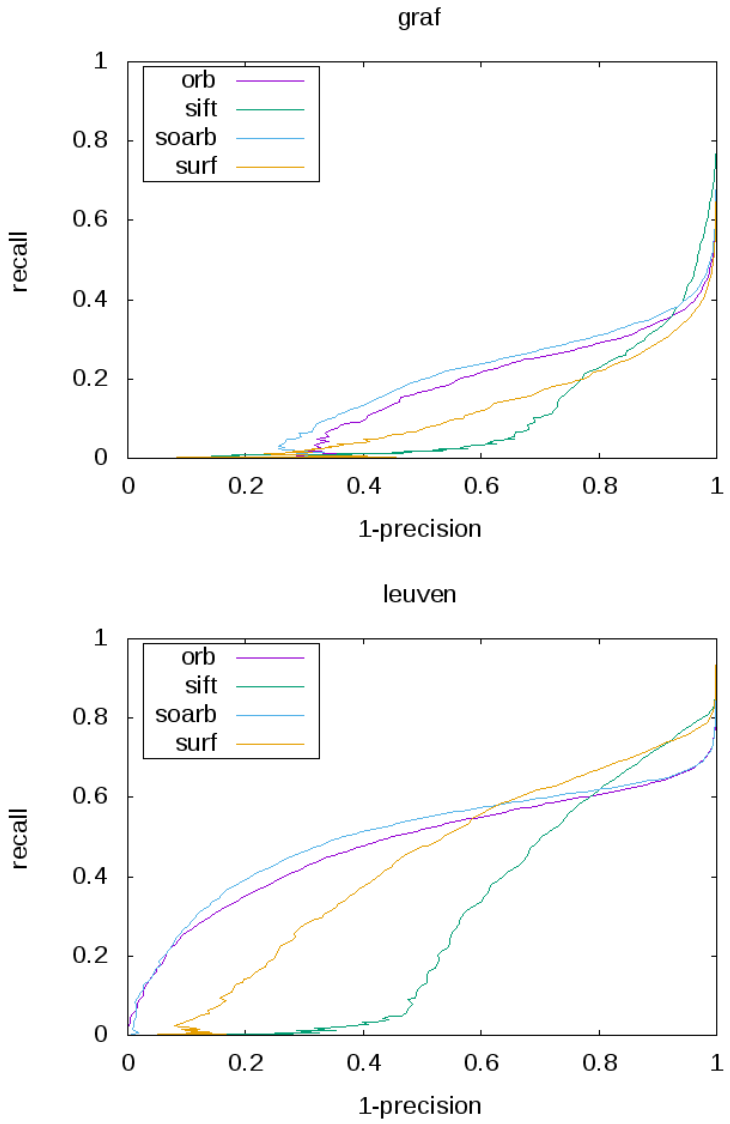


Abbildung 8.4: Oxford Deskriptor Evaluation - Recall- 1-Precision Kurven - Grafitti (Viewpoint) und Leuven (Light)

8.2 Evaluation der FPGA-Implementierung

Zur Evaluation der SOARB Implementierungen mit externen Beschleunigern wurden die Ausführungszeiten für Merkmalsdetektion und Merkmalsdeskription der GPU Implementierung mit der FPGA Implementierung verglichen. Die SOARB GPU-Implementierung wurde mit einer Nvidia GTX 1050Ti GPU getestet. Die Kernel für die FPGA-Implementierung wurden auf der Xilinx KCU1500 FPGA Beschleuniger-Karte ausgeführt. Beide getesteten Konfiguration nutzen einen Intel Core i7 6700K mit 3,6GHz als Host-System. Abbildung 8.5 zeigt das Test-System mit FPGA- und GPU-Beschleuniger.

Abbildung 8.5: Host-Computer mit KCU1500 Beschleuniger und Nvidia GTX1050Ti GPU

Tabelle 8.2 zeigt die Ausführungszeiten für 2000 Merkmale und die daraus resultierende Bildrate für beide Implementierungen. Bei den Zeiten für die Merkmalsdetektion ist die Berechnungszeit für die Erstellung der Skalenpyramiden enthalten.
Zusätzlich wurde bei den Tests die Leistungsaufnahme des Host-Systems über das Turbostat Tool, sowie die Leistungsaufnahme des jeweiligen Beschleunigers über das Nvidia-SMI Tool und den Xilinx Power Estimator (XPE) bestimmt. In Tabelle 8.3 sind die Leistungsaufnahmen beider Systeme und des Host-Systems aufgeführt.
Dabei ist der Leistungsverbrauch des Beschleunigermoduls (FPGA vs. GPU) um den Faktor 2,7x niedriger. Der Leistungsverbrauch des CPU Anteils ist

Implementierung	Det. Zeit(ms)	Deskr. Zeit(ms)	Total(ms)	FPS
SOARB (GPU)	9.32	6.62	15.94	62.73
SOARB (FPGA)	10.66	13.71	24.37	41.03

Tabelle 8.2: Evaluation der Implementierungen, Grafitti Datasets Image 1, 2000 Merkmale, Intel I7-6700K(3,6GHz) + GTX1050Ti vs. Intel I7-6700K(3,6GHz) + KCU1500

ebenfalls geringer. Die SOARB FPGA-Implementierung erreicht somit eine geringe Gesamt-Stromaufnahme von 32,23W. Dadurch bietet das System eine um Faktor 1,28x bessere Bildrate pro Watt (FPS/W) gegenüber der GPU-Implementierung.

Implementierung	Verbrauch CPU	Verbrauch Beschl.	Gesamt	FPS /W	Faktor
SOARB (GPU)	25.19W	38.0W	63.19W	0.992	1.0x
SOARB (FPGA)	18.53W	13.7W	32.23W	1.273	1.28x

Tabelle 8.3: Evaluation der Implementierungen, Leistungsaufnahme, Intel I7-6700K(3,6GHz) + GTX1050Ti vs. Intel I7-6700K(3,6GHz) + KCU1500

Tabelle 8.4 zeigt die verbrauchten Logikzellen und Ressourcen der FPGA-Implementierung auf dem Kintex Ultrascale XCKU115-2FLVB2104E FPGA des Xilinx KCU1500. Der SOARB Detektor-Kernel und Deskriptor-Kernel verbrauchen zusammen 26,5% der verfügbaren Logikzellen und 33% des Block-RAMs des XCKU115. Somit können in einem späteren Schritt noch weitere Funktionen wie z.B. eine Stereo-Bildverarbeitung nach Kapitel 3.5.1 mit in den FPGA integriert werden.

Ressource	**SOARB Detektor Kernel**	**SOARB Deskriptor Kernel**	**Verfügbar**
LUT	123479 (18.6%)	52737 (7.9%)	663360
FlipFlops	92434 (6.9%)	55015 (4.1%)	1326720
BlockRAM	1326 (30.7%)	102 (2.3%)	4320
DSP	5 (0.1%)	164 (3.0%)	5520
Frequency	180MHz	180MHz	

Tabelle 8.4: FPGA Auslastung des Xilinx KCU1500 (Kintex Ultrascale XCKU115-2FLVB2104E

8.3 SOARB Algorithmus in RTAB-Map

Zur Evaluation in einem merkmalsbasierten SLAM-System wurden der SOARB Algorithmus und der SOARB-RGB Algorithmus in RTAB-Map integriert. RTAB-Map unterstützt bereits eine große Zahl an Merkmals-Detektoren und Deskriptoren aus der OpenCV Bibliothek und bietet somit eine ideale Plattform zur Analyse und zum Vergleich der verschiedenen Algorithmen. Um eine gute Vergleichbarkeit der Ergebnisse zu gewährleisten, wurden Datasets für Odometrie und SLAM Anwendungen verwendet. Die KITTI-Datasets des Karlsruher Institute of Technologie (KIT) enthalten Einzelbildaufnahmen von gefahrenen Wegstrecken im Straßenverkehr und eine Referenz der Kamerabewegungen. Im nächsten Kapitel wird das KITTI Dataset vorgestellt. Die Kapitel 8.3.2 und 8.3.3 zeigen die Ergebnisse der Evaluation in den Datasets mit Graustufen-Bildern und Farbbildern.

8.3.1 KITTI Dataset

Das KITTI Odometrie Benchmark Dataset [GLU12] des Karlsruher Institute of Technologie (KIT) enthält 22 Stereo-Bildsequenzen von Szenarien im Straßenverkehr.
Die Datensätze beinhalten Aufnahmen von Stereo-Kameras in Graustufen und Farbe sowie Daten eines Velodyne-Laserscanners. Zu den Datensätzen 0 bis 10 sind ebenfalls die Ground Truth Posen als Referenz für Trainings- und Evaluationszwecke verfügbar. Abbildung 8.6 zeigt das Fahrzeug und die verwendete Sensorik, mit der die Datensätze aufgenommen wurden.

Abbildung 8.6: KITTI Odometrie Dataset - Fahrzeug mit Sensoren [GLU12]

Die Sequenzen sind eine Sammlung aller gängigen Szenarien im Straßenverkehr mit Fahrten in Städten und auf Landstraßen. Darin enthalten sind auch

andere Verkehrsteilnehmer wie z.B. Gegenverkehr und Fahrradfahrer. Diese stellen eine große Herausforderung für SLAM-Algorithmen dar, da diese Objekte meist einen anderen Bewegungsfluss als die Eigenbewegung erzeugen und somit die visuelle Odometrieberechnung erschweren.
Teilweise sind in den Sequenzen auch geschlossene Trajektorien und mehrfache Durchfahrten von Abschnitten enthalten, um die Detektion von Loop-Closures zu ermöglichen und zu analysieren. Dadurch bieten die Aufnahmen eine realistische Vergleichsmöglichkeit in Anwendungen der Odometrieberechnung, Kartographierung und Navigation für autonome Fahrzeuge und mobile Roboter.

Abbildung 8.7: KITTI Odometrie Dataset - Streckenübersicht [GLU12]

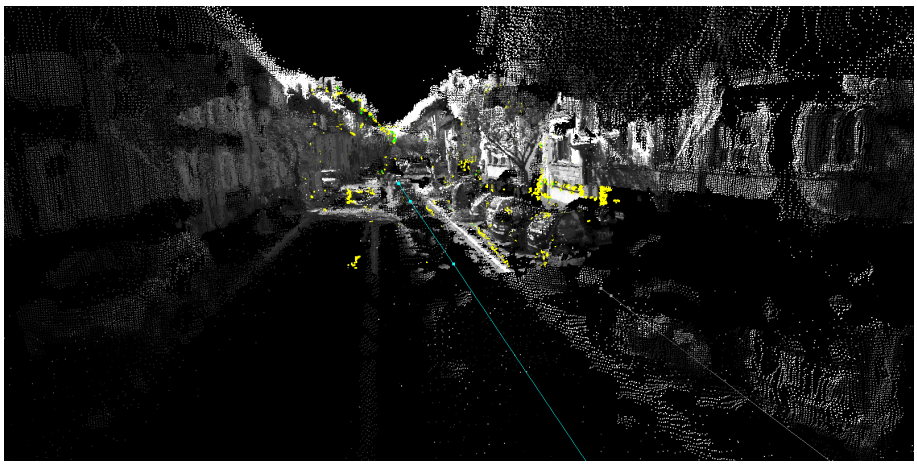
In Grafik 8.7 sind einige der Streckenverläufe dargestellt. Aufgrund der realistischen Umgebungen im Außenbereich und der Varianz der Daten mit wechselnden Beleuchtungen und Strukturen wurden die KITTI-Datasets für die Evaluation der Algorithmen im RTAB-System ausgewählt. Tabelle 8.5 gibt eine Übersicht der in dieser Arbeit verwendeten Datensätze 05 bis 10.

Datensatz	Streckenlänge	Szenario
05	2223m	Stadt
06	1239m	Stadt
07	695m	Stadt
08	3225m	Stadt + Landstraße
09	1717m	Stadt + Landstraße
10	919m	Stadt + Landstraße

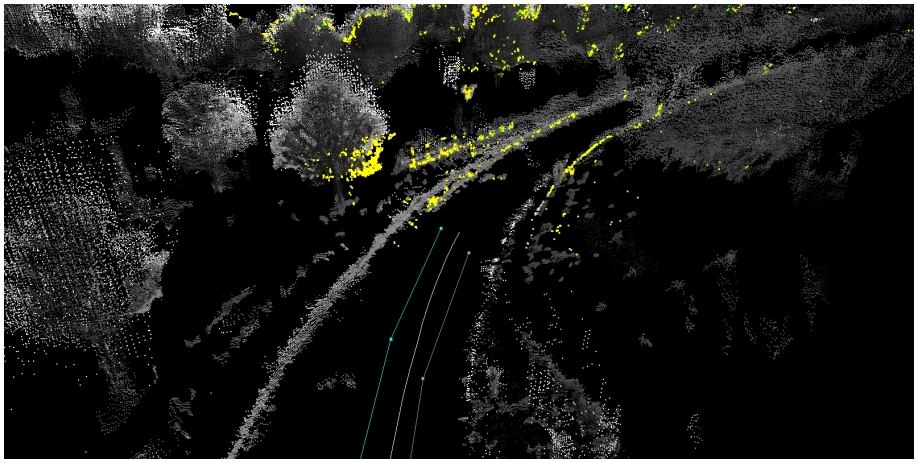
Tabelle 8.5: Verwendete KITTI-Datensätze

8.3.2 Evaluation im RTAB-Map SLAM System - Monochrome Kamera

Zur Analyse der SOARB-Merkmalserkennung wurden der SOARB-Detektor und der SOARB-Deskriptor in die RTAB-Map SLAM Umgebung eingebaut. Für die Tests und den Vergleich mit den anderen Verfahren wurden die Datensätze 05-10 ausgewählt und die SLAM-Kartographierung für alle zu testenden Verfahren durchgeführt. Dazu wurden alle Algorithmen mit gleiche Parametern für Odometrie und SLAM-Kartenerstellung in RTAB-Map konfiguriert. Die Anzahl der Merkmale wurde auf 2000 eingestellt.



(a)

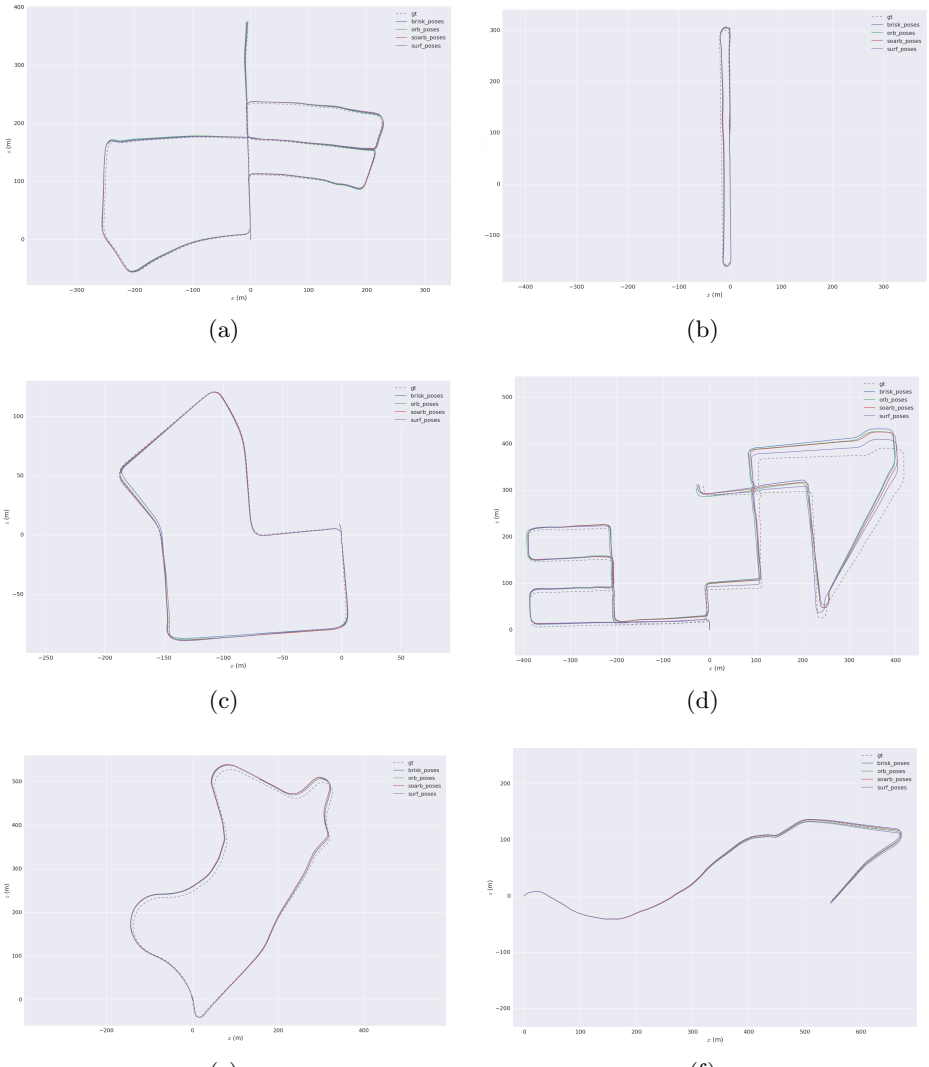


(b)

Abbildung 8.8: KITTI Dataset - 07(a) und 09(b) RTAB-Map SLAM 3D Karten aus SOARB Merkmalspunkten (Odometry + Mapping)

Abbildung 8.8 zeigt die generierten 3D-Ansichten der Sensordaten während

der Kartenerstellung aus den KITTI Datasets 07 und 09 unter Verwendung des SOARB-Verfahrens. Zur Evaluation und dem Vergleich mit anderen aktuellen Verfahren der Merkmalserkennung wurden neben dem vorgestellte SOARB-Algorithmus auch die Algorithmen BRISK, ORB und SURF getestet. Das SIFT-Verfahren wurde ebenfalls analysiert, hier konnten aber aufgrund einer großen Zahl verlorener Odometrie-Frames keinen verwendbaren Ergebnisse generiert werden.



(a) (b)

(c) (d)

(e) (f)

Abbildung 8.9: KITTI Dataset 05-10 RTAB-Map Ergebnis-Karten mit Ground-Truth (Vergrößerte Darstellung im Anhang)

Es wurden Karten aller Datensätze erstellt und die berechneten Bahnkurven zum Vergleich mit der Ground Truth Referenz gespeichert. Die Grafik 8.9

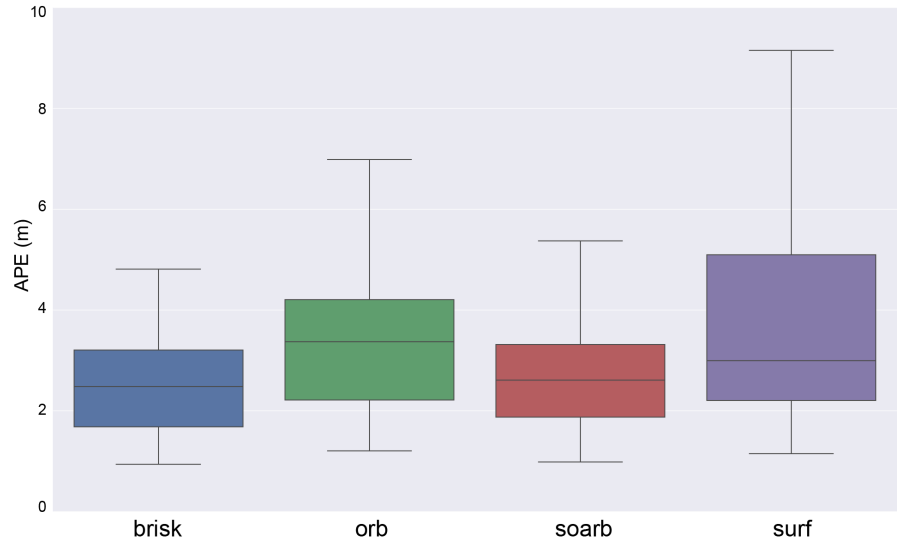
zeigt die 2D Karten aller Datensätze für die getesteten Algorithmen. Eine vergrößerte Darstellung der Karten ist im Anhang zu finden.
Tabelle 8.6 zeigt die Ergebnisse aller getesteten Verfahren für die KITTI-Datensätze 05 bis 10. In der Tabelle sind der Positionsfehler und der Rotationsfehler sowie der Root-Mean-Squared Error (RMSE) für Translation und Rotation aufgeführt. Die besten Ergebnisse sind jeweils grau hinterlegt. Zusätzlich wurden in der letzten Spalte die Anzahl der verlorenen Odometrie-Frames aufgeführt.

Data -set	Algorithm	t_err (%)	r_err (deg/m)	t_rmse (m)	r_rmse (deg)	Lost Frames
05	SOARB	0.860489	0.002483	1.915412	0.609033	0
	ORB	0.909602	0.002753	1.961543	0.646608	0
	SURF	0.799225	0.002579	1.642707	0.620492	1
	BRISK	0.823207	0.002767	1.765719	0.650065	0
06	SOARB	1.623143	0.003882	2.976354	0.792298	0
	ORB	1.983543	0.005119	3.667711	0.955495	1
	SURF	1.693203	0.003742	3.193185	0.795216	3
	BRISK	1.517374	0.004100	2.724646	0.812817	0
07	SOARB	0.599747	0.003816	0.558483	0.557733	0
	ORB	0.839037	0.004622	0.839643	0.582990	0
	SURF	0.662805	0.004367	0.488019	0.664405	1
	BRISK	1.002062	0.006210	1.347237	0.917704	0
08	SOARB	1.231775	0.003402	5.714197	1.742054	0
	ORB	1.364253	0.004423	7.762346	2.137867	1
	SURF	1.360834	0.004248	5.936282	2.011593	9
	BRISK	1.297923	0.003965	6.828977	2.097399	0
09	SOARB	1.349300	0.003007	3.986288	0.878924	0
	ORB	1.547630	0.004455	4.272648	0.968208	2
	SURF	1.347331	0.003244	4.214786	0.738765	5
	BRISK	1.297923	0.002920	3.863367	0.729753	0
10	SOARB	0.507947	0.002598	1.020449	0.594208	0
	ORB	0.647867	0.003999	1.287632	0.855922	1
	SURF	0.540750	0.002792	0.862940	0.503075	0
	BRISK	0.582849	0.003513	1.355794	0.849114	0
∅	SOARB	1.028734	0.003198	2.695197	0.862375	0.00
	ORB	1.215322	0.004228	3.298587	1.024515	0.83
	SURF	1,067358	0.003495	2.722987	0.888924	3.16
	BRISK	1,086890	0.003912	2.980957	1.009475	0.00

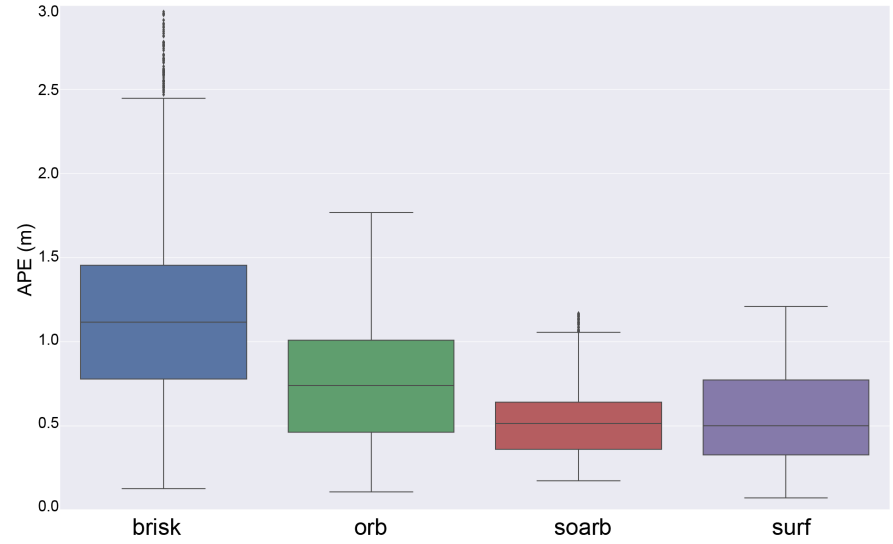
Tabelle 8.6: Evaluation der Implementierungen, KITTI Dataset 05-10

In der letzten Zeile der Tabelle 8.6 sind die mittleren Fehler über alle Datensätze berechnet.

Für eine präzise SLAM Lokalisierung und Kartenerstellung ist ein möglichst geringe Abweichung in Position und Rotation und ebenfalls ein möglichst geringe Anzahl an verlorener Odometrie-Frames (Lost Frames) wichtig.



(a)

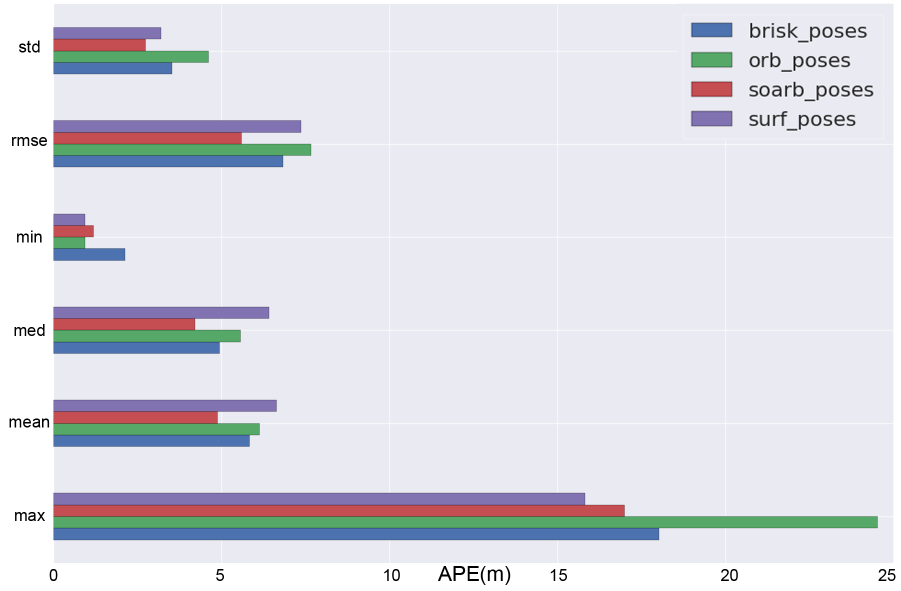


(b)

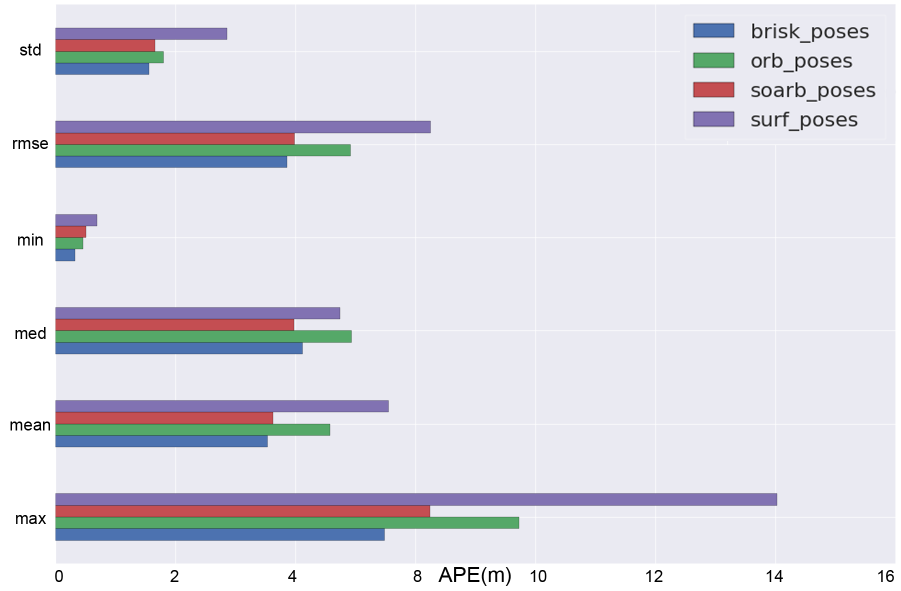
Abbildung 8.10: KITTI Dataset - 06(a) und 07(b) Absoluter Positionsfehler - Box Plots

Der SOARB Algorithmus erreicht in den meisten Datensatz-Benchmarks die geringsten Fehlerraten und verliert im Gegensatz zu den ORB und SURF

Algorithmen keine Odometrie-Frames. Das zeigt die gute Qualität der Merkmale und Deskriptoren in SLAM-Systemen.



(a)



(b)

Abbildung 8.11: KITTI Dataset - 08(a) und 09(b) Absoluter Positionsfehler - Statistiken

Im Durchschnitt über alle Datensätze erreicht das vorgestellte SOARB-

Verfahren die besten Ergebnisse aller evaluierten Algorithmen. SOARB erzielt durchschnittlich eine Verbesserung des Translation-RMSE um 22% und eine Verbesserung Rotations-RMSE von 19% im Vergleich zu ORB. Die Maximale Steigerung im Vergleich zu ORB lag bei 50% Translation-RMSE und 44% Rotation-RMSE. Der RMSE ist dabei ein guter Vergleichswert in der Evaluation von SLAM-Systemen, da er Ausreißer in den Messungen höher gewichtet als z.B. der Median oder der Mittelwert der Fehler.
Mit dem Python Tool evo [Gru18] von Michael Grupp wurden Trajektorien zusätzlich ausgewertet und visualisiert. Bild 8.10 zeigt die absoluten Positionsfehler der getesteten Verfahren für die KITTI Odometrie-Datensätze 06 und 07. Der SOARB Algorithmus erreicht dabei einen deutlich geringeren Fehler im Vergleich zu dem ORB-Verfahren. Auch im Vergleich mit den rechenaufwändigeren Merkmalserkennungsverfahren BRISK und SURF können verbesserte bzw. gleichwertige Ergebnisse erzielt werden. In den Grafiken 8.11 a) und b) sind die Statistiken des absoluten Positionsfehlers für die KITTI Odometrie-Datensätze 08 und 09 als Balkendiagramm dargestellt. Hier sind neben dem RMSE-Wert und der Standardabweichung auch der mittlere Fehler, der Medianwert sowie der minimale und maximale Fehler gezeigt.
Der absolute Positionsfehler über den Verlauf der Strecke von dem KITTI-Datensatz 07 ist in Bild 8.12 visualisiert.

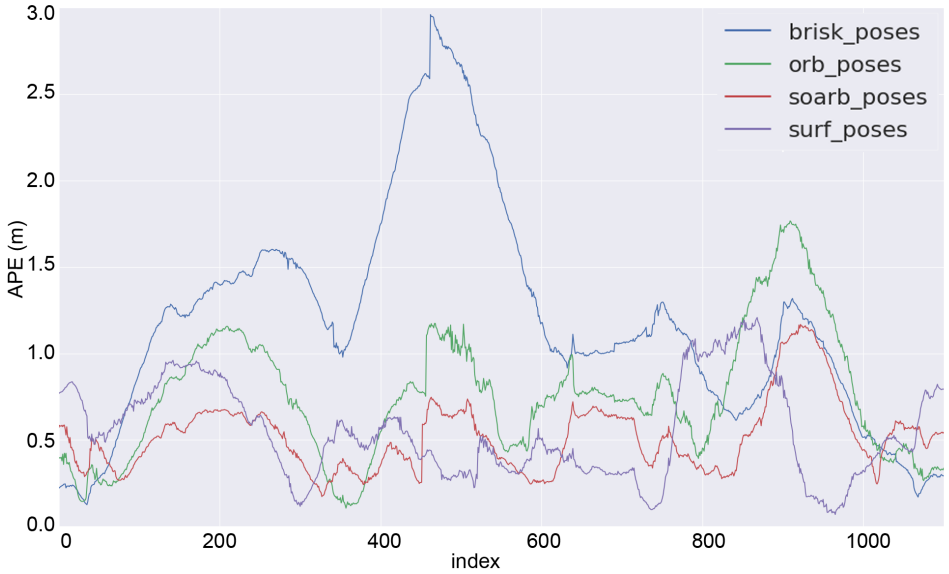


Abbildung 8.12: Absolut Position Error(APE) - Verlauf über Fahrstrecke, Dataset 07

8.3.3 Evaluation im RTAB-Map SLAM System - Farbkamera

In diesem Kapitel soll der Einfluss der Verwendung von Farbinformationen aus einem Datensatz mit Stereo-Farbbildern untersucht werden. Das RTAB-Map System wurde dazu modifiziert, um Bilder mit den drei Farbkanälen Rot, Grün und Blau zu verarbeiten. Die eingelesenen Datensatz-Bilder werden in Farbbilder an die interne Merkmalsverarbeitung in der Odometrie und Kartenerstellung weitergereicht. Alle weiteren Komponenten, wie auch die herkömmlichen Merkmalsdetektoren und Deskriptoren nutzen weiterhin eine Graustufen-Konvertierung der Eingangsbilder.

Abbildung 8.13: RTAB-Map 3D Karten aus SOARB Merkmalspunkten, Dataset 07, RGB-Daten

Dadurch ist eine gute Vergleichsmöglichkeit des SOARB-RGB-Deskriptors mit den anderen Verfahren, und dem SOARB-Deskriptor mit monochromen Daten, gegeben. Zur Evaluation der Implementierung mit dem erweiterten Farb-Deskriptor wird in diesem Kapitel der KITTI-Datensatz 07 verwendet. Zusätzlich zu dem SOARB-RGB Verfahren werden SOARB, ORB und SURF betrachtet. Abbildung 8.14 zeigt die Bahnkurven aus dem RTAB-Map SLAM mit den verschiedenen Verfahren.
In Tabelle 8.7 sind der Translationsfehler und der Rotationsfehler dargestellt. Die Daten zeigen, dass für den KITTI-Datensatz 07 mit RGB-Daten der SOARB-RGB Deskriptor den Translationsfehler weiter verringern kann. Gegenüber der ORB-Implementierung erreicht SOARB-RGB eine Verbesserung des Translationsfehlers und Rotationsfehlers von über 45%.
Im Vergleich zu SURF bieten beide vorgestellten Implementierungen einen geringeren Rotationsfehler. Nur in der Translation erzielt der aufwändigere

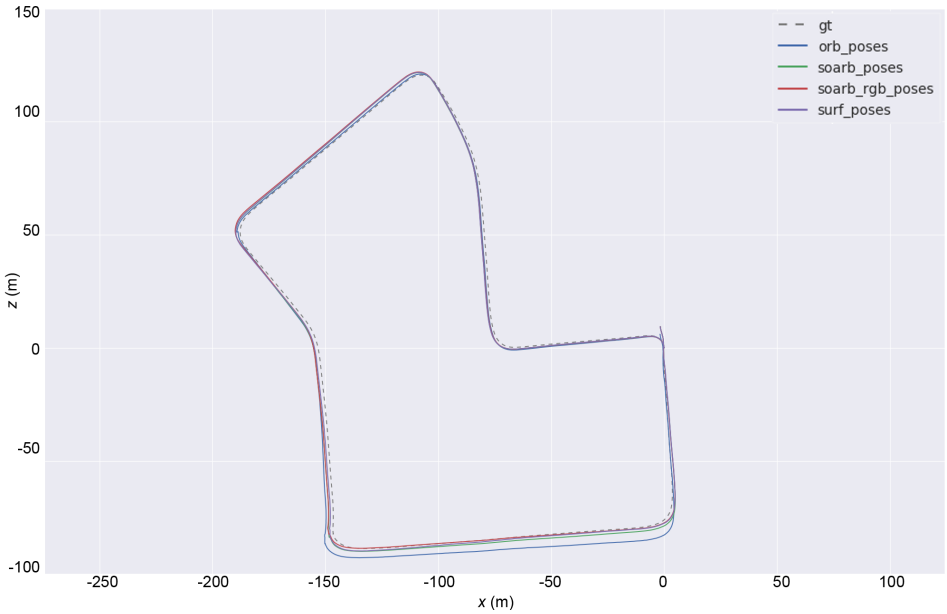


Abbildung 8.14: Karte aus Stereo-Dataset, Dataset 07, RGB-Daten

Implementierung	Translation Error	Rotation Error
SOARB-RGB (vorgestellt)	1.04 (%)	0.00387 deg/m
SOARB (vorgestellt)	1.09 (%)	0.00349 deg/m
ORB	1.51 (%)	0.00567 deg/m
SURF	0.98 (%)	0.00400 deg/m

Tabelle 8.7: Evaluation der Implementierungen in RTAB-Map, KITTI 07 Dataset mit RGB-Kameras

SURF-Algorithmus ein besseres Ergebnis.
Darüberhinaus wurde mit dem evo Tool [Gru18] der Positionsfehler analysiert. Grafik 8.15 zeigt die Positionsabweichungen in X-, Y- und Z-Richtung. In den Abbildungen 8.16 und 8.17 ist der absolute Positionsfehler als Box-Plots bzw. Diagramm über die Wegstrecke visualisiert.
Auch hier zeigt der SOARB-RGB Deskriptor eine Verbesserung im Vergleich zu ORB und SOARB. Allerdings zeigen die Tests in dem RTAB-Map-System bei den Datensätzen mit realen Beleuchtungen und Bildfolgen, dass hier die zusätzlichen Farbinformation des SOARB-RGB-Deskriptors nur weniger zum Tragen kommen als z.B. im Oxford Datensatz. Daher kann für SLAM-Anwendungen hier zwar eine leichte Verbesserung im Vergleich zu dem normalen SOARB-Deskriptor erreicht werden, es konnte aber eine generell höhere Fehlerrate aller Algorithmen durch die Verwendung der RGB-Bilder beobachtet werden.

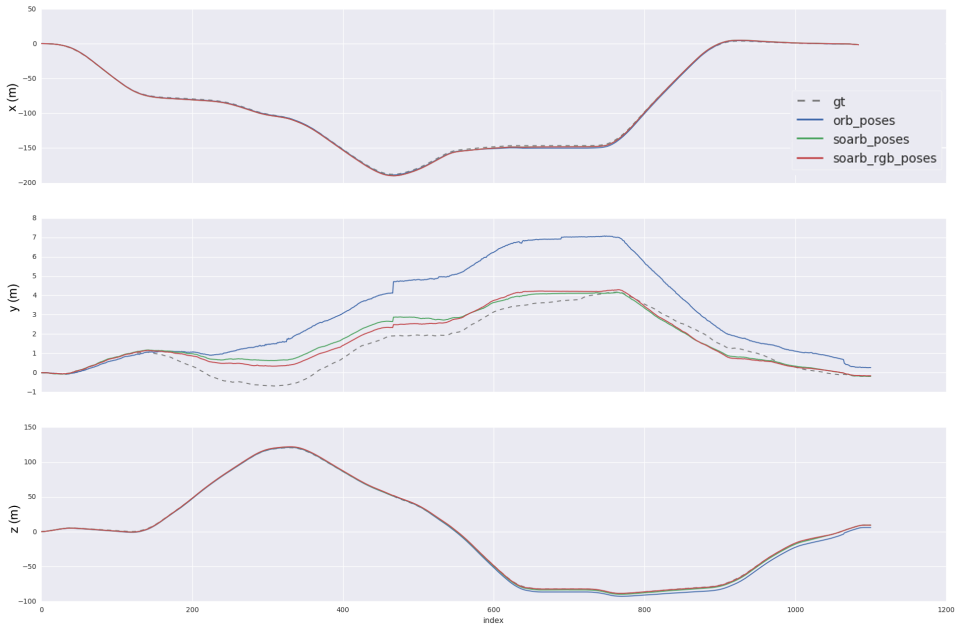


Abbildung 8.15: XYZ Verlauf, Dataset 07, RGB-Daten

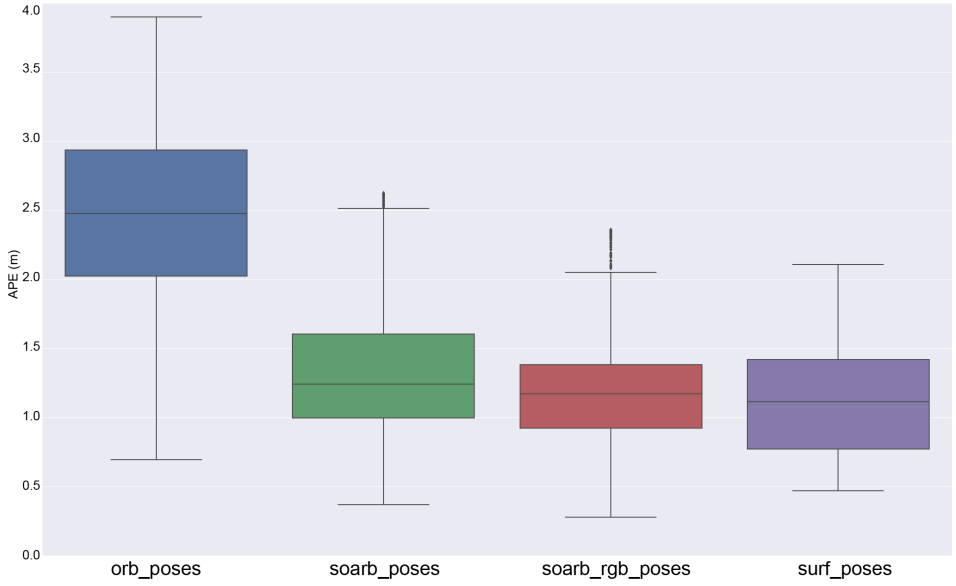


Abbildung 8.16: Absolut Position Error(APE) - Box-Plot, Dataset 07, RGB-Daten

Somit ist der höhere Implementierungsaufwand durch die Farbinformationen im Deskriptor und die doppelte Deskriptorlänge nur in Anwendungen mit hohen Farbtiefen bzw. Farbdifferenzen zielführend. Für die Implemen-

Abbildung 8.17: Absolut Position Error(APE) - Diagramm, Dataset 07, RGB-Daten

tierung in dem FPGA-Beschleuniger wurde daher der SOARB-Deskriptor mit Graustufen-Informationen ausgewählt.
Eine größere Darstellung der in diesem Kapitel gezeigten Diagramme ist im Anhang zu finden.

Kapitel 9

Zusammenfassung und Ausblick

In dieser Arbeit wurden Methoden zur Optimierung einer Merkmalserkennung für visuelle merkmalsbasiserte SLAM Verfahren mit Hardware-Beschleunigern untersucht. Dazu wurden Verfahren zur Subpixel-genauen Bestimmung der Merkmalsposition analysiert und zusätzlich eine Erweiterung der Merkmalsdeskriptoren mit Farbinformationen und Subpixel-genauer Rotation der Deskriptor-Pattern betrachtet. Die untersuchten Verfahren wurden hinsichtlich ihrer Eigenschaften und der Implementierbarkeit auf einem Hardware-Beschleuniger evaluiert und ausgewählt.

Aus den Ergebnissen der Untersuchung konnte ein für SLAM-Anwendungen angepasster Subpixel-accurate Oriented AGAST and Rotated BRIEF (SOARB) Algorithmus entworfen werden. Der Algorithmus weist trotz der effizienten und Ressourcen-optimierten Implementierung eine Verbesserung der Merkmalsdetektion und -Deskription in Relation zu anderen vergleichbaren Verfahren auf.

In der Arbeit wurde das SOARB-Verfahren mithilfe der Xilinx SDAccel Umgebung für Hardware-Software-Codesign in ein SLAM System mit KCU1500 FPGA-Beschleuniger integriert. Dazu wurden OpenCL Kernel-Module des Detektors und des Deskriptors, sowie ein OpenCL-Framework zur Anbindung an ein merkmalsbasiertes SLAM-System erstellt.

Die FPGA-Implementierung des SOARB-Verfahrens erreicht dabei eine Bildrate von 41 Bildern/s und ist damit um Faktor 2,6x schneller als AKAZE, die schnellste GPU-basierte Implementierung in OpenCV mit Subpixel-genauer Bestimmung der Merkmalsposition. Im Vergleich zu SIFT und SURF wird eine Beschleunigung der Ausführung von Faktor 5,3x bzw. 4,7x erreicht.

Dabei bietet die SOARB FPGA-Implementierung eine geringe Leistungsaufnahme von 13,7W (FPGA). Im Vergleich zu der ebenfalls erstellten SOARB GPU-Referenzimplementierung kann eine 1,28x höhere Leistungseffizienz (Bilder/s pro Watt) erreicht werden.

Zur Evaluation wurde das gezeigte Verfahren in dem RTAB-Map SLAM-System mit Datensätzen zur Bewertung der Lokalisierung und Kartenerstellung analysiert und mit anderen State-of-the-Art Algorithmen verglichen. Die Untersuchung des Beschleuniger-Systems wurde dabei anhand von den KITTI-Datensätzen mit Bildaufnahme-Sequenzen aus dem Straßenverkehr durchgeführt. Diese lassen sich durch die Art der Aufnahmen sehr gut auf reale Anwendungen der SLAM-Navigation von autonomen Fahrzeugen übertragen.
Der SOARB-Algorithmus erzielt in den Tests eine Verbesserung des Translations- und Rotationsfehlers von durchschnittlich 22% und 19% im Vergleich zu dem ORB-Verfahren. Die maximale Verbesserung lag bei 50% des Translations-RMSE bzw. 40% des Rotations-RMSE.
Darüber hinaus wurde die Merkmalserkennung des RTAB-Map Systems um Farbinformationen erweitert, um auch den präsentierten SOARB-RGB Algorithmus als GPU-Implementierung in einer SLAM-Anwendung zu evaluieren. Hier wurde eine zusätzliche Verbesserung von 4,8% im Vergleich zu SOARB erzielt. Bei dem SOARB-RGB Deskriptor wurde eine hohe Abhängigkeit von dem verwendeten Datensatz (Oxford vs. KITTI) festgestellt. Aufgrund des höheren Implementierungsaufwands durch die Farbinformationen und des dadurch höheren Ressourcenverbrauchs sollte der Aufwand einer Hardware-Implementierung daher abhängig von der Anwendung bewertet werden.
In dieser Arbeit wurde daher der SOARB-Deskriptor mit Graustufen-Informationen für die Hardware-Implementierung gewählt.

9.1 Ausblick

Diese Arbeit hat die Vorteile der HW-SW-Codesign-Methodik der Xilinx SDAccel Umgebung zur Implementierung einer Merkmalserkennung gezeigt. Ein Vorzug des verwendeten FPGA-Beschleunigers ist, dass mehrere Beschleuniger-Module parallel ausgeführt werden können und untereinander sogar Daten austauschen können.
Das SOARB-Verfahren nutzt derzeit nur ca. 33% der verfügbaren Ressourcen des KCU1500 Beschleunigers. Daher könnte in einer weiterführenden Arbeit eine Stereo-Bildverarbeitung wie in Kapiteln 3.5.1 und 7.2 beschrieben in den Beschleuniger integriert werden.
Das SDAccel Tool bietet dazu einen RTL-Kernel-Wizzard zur Einbindung von VHDL und Verilog Modulen in das SDAccel-System. Somit können die HDL-Module an die AXI-Schnittstellen angebunden und direkt in OpenCL als Beschleuniger verwendet werden.
Durch die Integration der SRCC Stereo-Tiefenberechnung oder eines ähnlichen Verfahrens könnte die SLAM Lokaliserung und Kartenerstellung zusätzlich verbessert und die Fehler der visuellen Odometrie reduziert werden.

9.2 Danksagung

Ich möchte mich ganz besonders bei den beiden betreuenden Professoren Prof. Dr. Mladen Berekovic und Prof. Dr. Jochen Steil bedanken, die mich in meiner bisherigen Zeit als wissenschaftlicher Mitarbeiter der TU Braunschweig begleitet und unterstützt haben.
Ich danke Prof. Dr. Mladen Berekovic, dass er mir mit seinen Erfahrungen im Bereich des ASIC-Designs und der Entwurfsmethodik weitergeholfen hat und damit zur fachlichen Ausrichtung der vorliegenden Arbeit beigetragen hat. Prof. Dr. Jochen Steil möchte ich für die fachliche Zusammenarbeit im Themengebiet der Robotik und für seine Hilfe bei Fragen zum Thema der Promotion danken.
Nicht zuletzt danke ich meiner Familie und meiner Frau Vanessa, die mich auf meinem Weg zur Promotion begleitet haben und ohne deren Unterstützung diese Arbeit nicht möglich gewesen wäre.

Abbildungsverzeichnis

Literaturverzeichnis

[AMB11] ARM AMBA. 3.0 axi specification, 2011.

[AOV12] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. Freak: Fast retina keypoint. In *Computer vision and pattern recognition (CVPR), 2012 IEEE conference on*, pages 510–517. Ieee, 2012.

[AS11] Pablo F Alcantarilla and T Solutions. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *IEEE Trans. Patt. Anal. Mach. Intell*, 34(7):1281–1298, 2011.

[Bac] FH-Prof. DI Dr. Werner Backfrieder. Vorlesungsskript fortgeschrittene bildverarbeitung & analyse (fba). Research report, Institut für Informatik, FH-Hagenberg.

[Ber10] Prof. Dr.-Ing. Mladen Berekovic. Vorlesungsskript advanced vlsi design 2. Research report, C3E, Technische Universität Braunschweig, 2010.

[Bis04] Dr. Kanad Biswas. Vorlesungsskript computer vision. Research report, Computer Science, University of Central Florida, 2004.

[BL02] Matthew Brown and David G Lowe. Invariant features from interest point groups. In *BMVC*, volume 4, 2002.

[BMC08] Vanderlei Bonato, Eduardo Marques, and George A Constantinides. A parallel hardware architecture for scale and rotation invariant feature detection. *IEEE transactions on circuits and systems for video technology*, 18(12):1703–1712, 2008.

[Bra13] Anselm Brachmann. Erweiterung von fast retina keypoints um farbinformation. Master's thesis, FU Berlin, 2013.

[BTVG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[CB18] Nicholas Carlevaris-Bianco. Perl next generation vehicle. Website, 2018. Available online at `http://www.nickcb.com/` visited on Aug 21th 2018.

[CLSF10] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[DMEW02] Giovanni De Micheli, Rolf Ernst, and Wayne Wolf, editors. *Readings in hardware/software co-design*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[FA13] Wade S Fife and James K Archibald. Improved census transforms for resource-optimized stereo vision. *IEEE Transactions on Circuits and Systems for Video Technology*, 23(1):60–73, 2013.

[FWR10] Udo Frese, René Wagner, and Thomas Röfer. A slam overview from a user?s perspective. *KI-Künstliche Intelligenz*, 24(3):191–198, 2010.

[FZYL17] Weikang Fang, Yanjun Zhang, Bo Yu, and Shaoshan Liu. Fpga-based orb feature extraction for real-time visual slam. In *Field Programmable Technology (ICFPT), 2017 International Conference on*, pages 275–278. IEEE, 2017.

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[Gru18] Michael Grupp. Evo - python package for the evaluation of odometry and slam. Website, 2018. Available online at `https://github.com/MichaelGrupp/evo` visited on Aug 15th 2018.

[Gui18] Erico Guizzo. How google's self-driving car works. Website, 2018. Available online at `https://spectrum.ieee.org/automaton/robotics/artificial-intelligence/how-google-self-driving-car-works` visited on Aug 21th 2018.

[GZS11] Andreas Geiger, Julius Ziegler, and Christoph Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *Intelligent Vehicles Symposium (IV)*, 2011.

[HB86] Kai Hwang and Faye A. Briggs. *Computer architecture and parallel processing.* McGraw-Hill Series in computer organization and architecture. McGraw-Hill, 1986.

[HBH+17] Albert S Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *Robotics Research*, pages 235–252. Springer, 2017.

[HCM15] Yuanfeng Han, Peijiang Chen, and Tian Meng. Harris corner detection algorithm at sub-pixel level and its application. *Adv. Comput. Sci. Res*, 2015.

[HP94] John L. Hennessy and David A. Patterson. *Rechnerarchitektur - Analyse, Entwurf, Implementierung, Bewertung.* Vieweg Lehrbuch Informatik. Vieweg, 1994.

[HP03] John Hennessy and David Patterson. *Computer Architecture - A Quantitative Approach.* Morgan Kaufmann, 2003.

[HS88] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[Jäh05] Bernd Jähne. *Digitale Bildverarbeitung.* Springer Verlag, 6., überarb. u. erw. aufl. edition, April 2005.

[KG17] Lester Kalms and Diana Göhringer. Exploration of opencl for fpgas using sdaccel and comparison to gpus and multicore cpus. In *Field Programmable Logic and Applications (FPL), 2017 27th International Conference on*, pages 1–4. IEEE, 2017.

[KM08] Georg Klein and David Murray. Improving the agility of keyframe-based slam. In *European Conference on Computer Vision*, pages 802–815. Springer, 2008.

[KPS17] Ebrahim Karami, Siva Prasad, and Mohamed Shehata. Image matching using sift, surf, brief and orb: performance comparison for distorted images. *arXiv preprint arXiv:1710.02726*, 2017.

[KSC13] Christian Kerl, Jurgen Sturm, and Daniel Cremers. Dense visual slam for rgb-d cameras. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 2100–2106. Citeseer, 2013.

[Lab18] Mathieu Labbe. Rtab-map ros wiki - stereo outdoor navigation. Website, 2018. Available online at `http://wiki.ros.org/rtabmap_ros/Tutorials/StereoOutdoorNavigation` visited on Aug 15th 2018.

[LCS11] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011.

[LK+81] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.

[LM11] Mathieu Labbé and François Michaud. Memory management for real-time appearance-based loop closure detection. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1271–1276. IEEE, 2011.

[LM13] Mathieu Labbe and Francois Michaud. Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE Transactions on Robotics*, 29(3):734–745, 2013.

[LM14] Mathieu Labbe and François Michaud. Online global loop closure detection for large-scale multi-session graph-based slam. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2661–2666. IEEE, 2014.

[Low99] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[Low04] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[MAMT15] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[MAT17] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[MHB+10] Elmar Mair, Gregory D Hager, Darius Burschka, Michael Suppa, and Gerhard Hirzinger. Adaptive and generic corner de-

tection based on the accelerated segment test. In *European conference on Computer vision*, pages 183–196. Springer, 2010.

[Mic10] Sören Michalik. Entwurf eines Stereo-Bildverarbeitungssystems im Roboterfußball, 2010.

[Mic12] Sören Michalik. Entwicklung eines Leon3-Softcore-Systems mit rekonfigurierbarem Bildverarbeitungsprozessor, 2012.

[MMNB17] Soenke Michalik, Soeren Michalik, Jamin Naghmouchi, and Mladen Berekovic. Real-time smart stereo camera based on fpga-soc. In *Humanoid Robotics (Humanoids), 2017 IEEE-RAS 17th International Conference on*, pages 311–317. IEEE, 2017.

[Mol05] Prof. Dr. Paul Molitor. Vorlesungsskript technische informatik. Research report, Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg, 2005.

[MS05] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE transactions on pattern analysis and machine intelligence*, 27(10):1615–1630, 2005.

[MTS+05] Krystian Mikolajczyk, Tinne Tuytelaars, Cordelia Schmid, Andrew Zisserman, Jiri Matas, Frederik Schaffalitzky, Timor Kadir, and Luc Van Gool. A comparison of affine region detectors. *International journal of computer vision*, 65(1-2):43–72, 2005.

[Nav09] Prof. Dr. Nassir Navab. Vorlesungsskript 3d computer vision - stereo matching. Research report, Institut für Informatik, Technische Universität München, 2009.

[Ope18] OpenRobots. Ros navigation tutorial. Website, 2018. Available online at `http://www.openrobots.org/morse/doc/1.4/user/advanced_tutorials/ros_nav_tutorial.html` visited on Aug 13th 2018.

[OYM+16] Takeshi Ohkawa, Kazushi Yamashina, Takuya Matsumoto, Kanemitsu Ootsu, and Takashi Yokota. Architecture exploration of intelligent robot system using ros-compliant fpga component. In *Proceedings of the 27th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype*, pages 72–78. ACM, 2016.

[QCG+09] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[RD06] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[Ros06] Jean-Claude Rosenthal. Feature Detection und Matching Verfahren zur Positionund Lagebestimmung, 2006.

[RRKB11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[Ste08] Johannes Steinmüller. *Bildanalyse: Von der Bildverarbeitung zur räumlichen Interpretation von Bildern (eXamen.Press)*. Springer Verlag, 1., ed. edition, August 2008.

[SZC+18] Amr Suleiman, Zhengdong Zhang, Luca Carlone, Sertac Karaman, and Vivienne Sze. Navion: A fully integrated energy-efficient visual-inertial odometry accelerator for autonomous navigation of nano drones. 2018.

[TAA16] Siok Yee Tan, Haslina Arshad, and Azizi Abdullah. A new illumination invariant feature based on freak descriptor in rgb color space. *Journal of Theoretical & Applied Information Technology*, 93(1), 2016.

[Tan06] Andrew S. Tanenbaum. *Computerarchitektur - Strukturen, Konzepte, Grundlagen (5. ed.)*. Pearson Education, 2006.

[TK91] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. 1991.

[uMS12] Marc Reichenbach und Michael Schmidt. Vhdl - cordic verfahren, informatik 3 / rechnerarchitektur. Presentation, 2012. Universität Erlangen Nürnberg.

[Vol59] Jack E Volder. The cordic trigonometric computing technique. *IRE Transactions on electronic computers*, (3):330–334, 1959.

[Wal71] John S Walther. A unified algorithm for elementary functions. In *Proceedings of the May 18-20, 1971, spring joint computer conference*, pages 379–385. ACM, 1971.

[WKBD17] Josh Weberruss, Lindsay Kleeman, David Boland, and Tom Drummond. Fpga acceleration of multilevel orb feature extraction for computer vision. In *Field Programmable Logic and Applications (FPL), 2017 27th International Conference on*, pages 1–8. IEEE, 2017.

[WKD15] Josh Weberruss, Lindsay Kleeman, and Tom Drummond. Orb feature extraction and matching in hardware. In *Proceedings of the Australasian Conference on Robotics and Automation, the Australian National University, Canberra, Australia*, pages 2–4, 2015.

[xila] *KCU1500 User Guide.*

[xilb] Sdaccel backgrounder. Technical report.

[xilc] *SDAccel Environment Profiling and Optimization Guide.*

[xild] *SDAccel Environment User Guide.*

[XZZX13] Yuanxiu Xing, Dengyi Zhang, Jianhui Zhao, and Aiping Xu. A fast and reliable corner detector for non-uniform illumination mineshaft images. *Journal of Intelligent Systems*, 22(4):453–470, 2013.

[YBHH15] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. An overview to visual odometry and visual slam: Applications to mobile robotics. *Intelligent Industrial Systems*, 1(4):289–311, 2015.

[YDMJ18] Yi Yang, Fajie Duan, Ling Ma, and Jiajia Jiang. A robust method for constructing rotational invariant descriptors. *Signal Processing: Image Communication*, 60:224–236, 2018.

[YMOFE17] Xin Yuan, José-Fernán Martínez-Ortega, José Antonio Sánchez Fernández, and Martina Eckert. Aekf-slam: a new algorithm for robotic underwater navigation. *Sensors*, 17(5):1174, 2017.

[ZWW07] Qing Zhu, Bo Wu, and Neng Wan. A sub-pixel location method for interest points by means of the harris interest strength. *The Photogrammetric Record*, 22(120):321–335, 2007.

Kapitel 10

Anhang

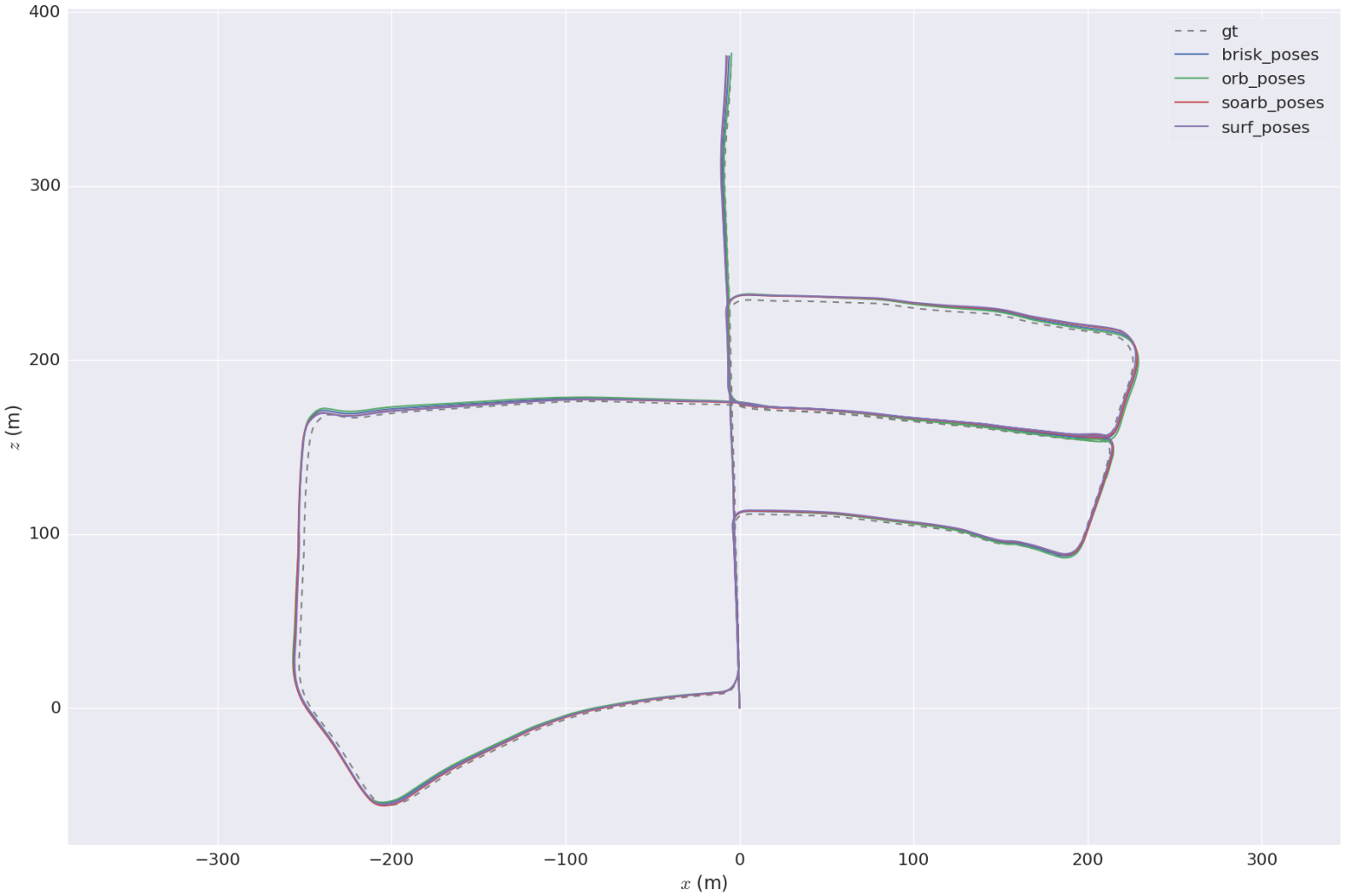


Abbildung 10.1: KITTI Dataset 05 - RTAB-Map Ergebnis-Karte mit Ground-Truth

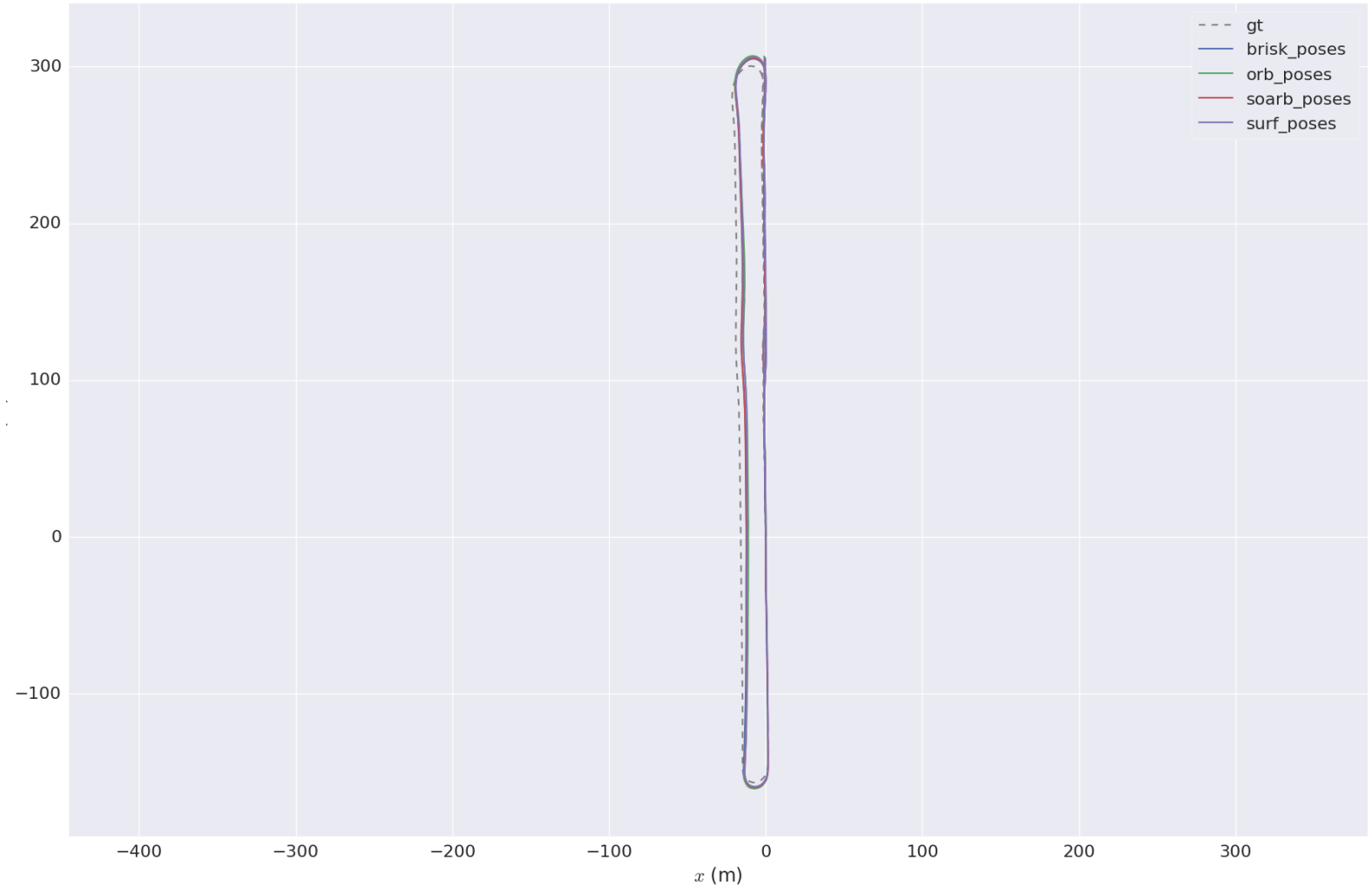


Abbildung 10.2: KITTI Dataset 06 - RTAB-Map Ergebnis-Karte mit Ground-Truth

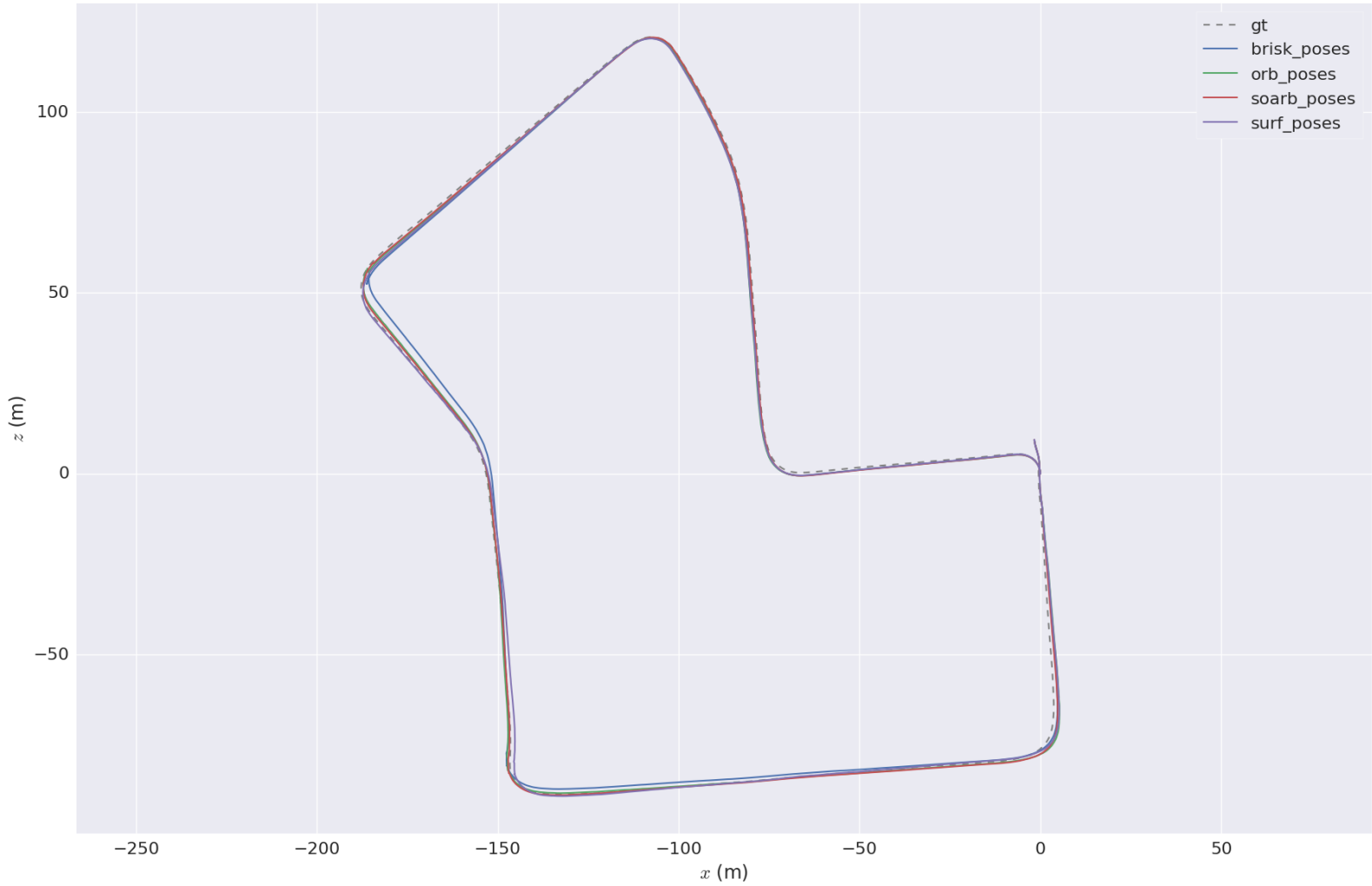


Abbildung 10.3: KITTI Dataset 07 - RTAB-Map Ergebnis-Karte mit Ground-Truth

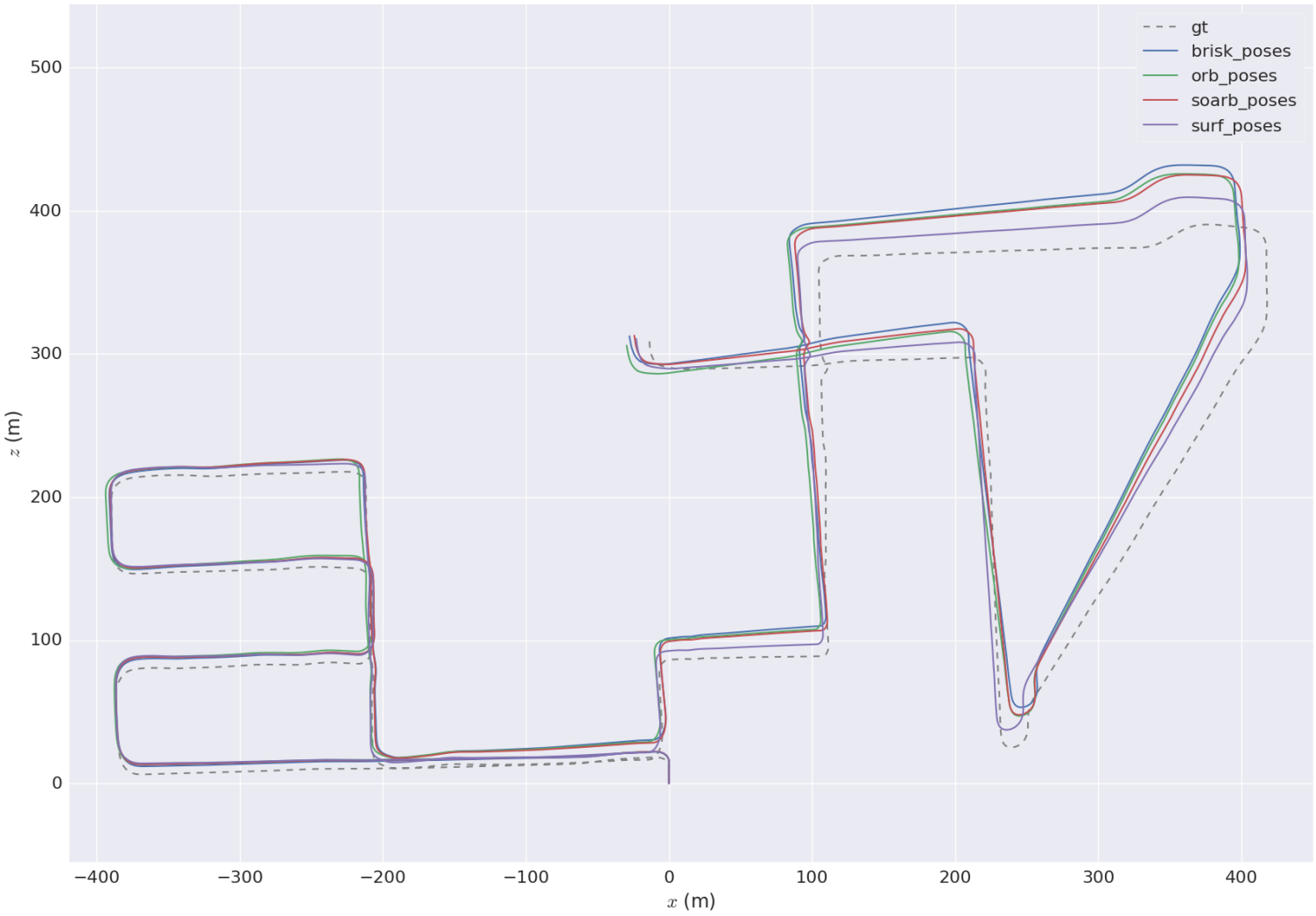


Abbildung 10.4: KITTI Dataset 08 - RTAB-Map Ergebnis-Karte mit Ground-Truth

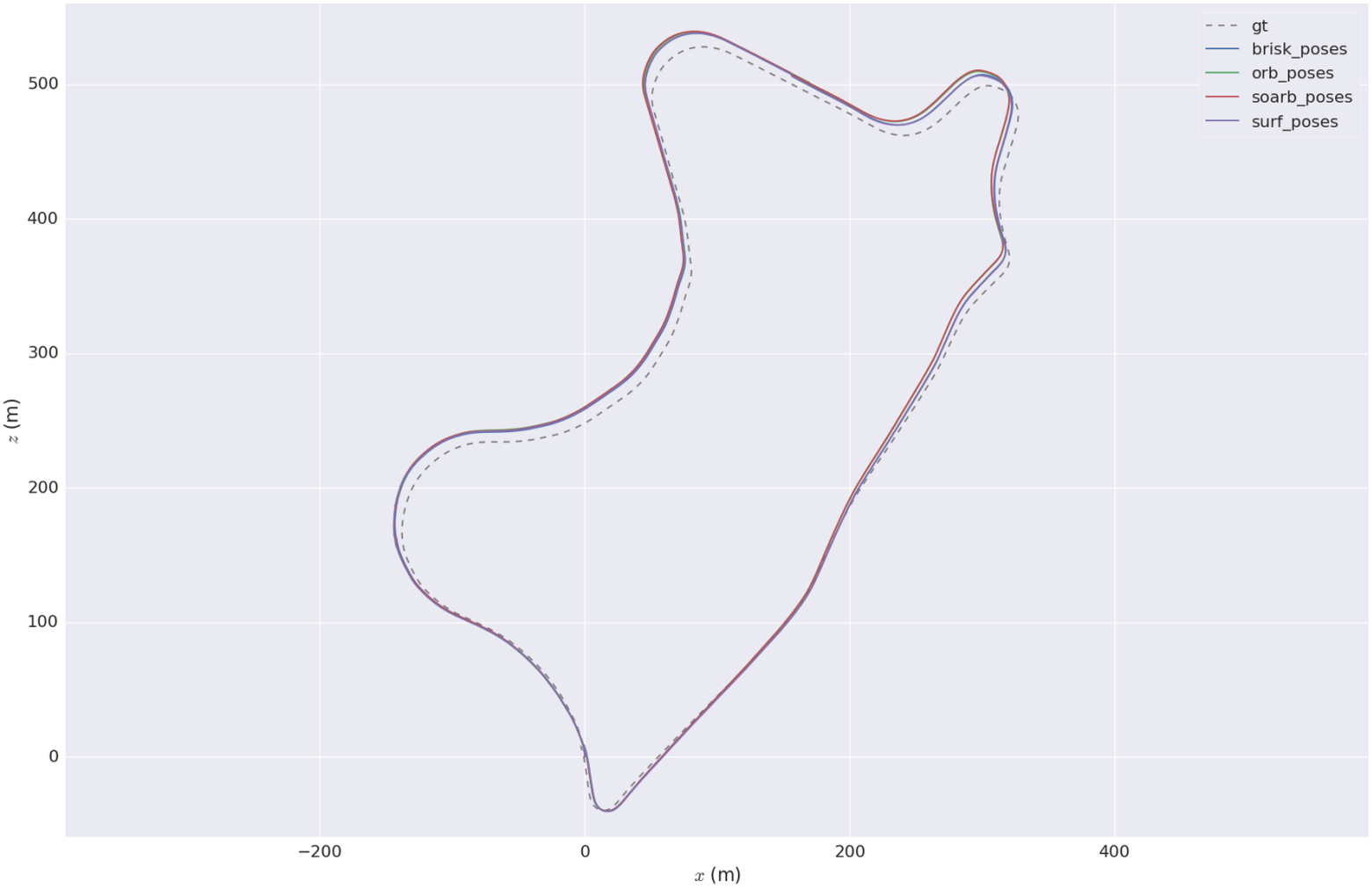


Abbildung 10.5: KITTI Dataset 09 - RTAB-Map Ergebnis-Karte mit Ground-Truth

Abbildung 10.6: KITTI Dataset 10 - RTAB-Map Ergebnis-Karte mit Ground-Truth

Abbildung 10.7: KITTI Dataset 07 - RTAB-Map Ergebnis-Karte mit Ground-Truth (RGB-Datensatz / SOARB-RGB)

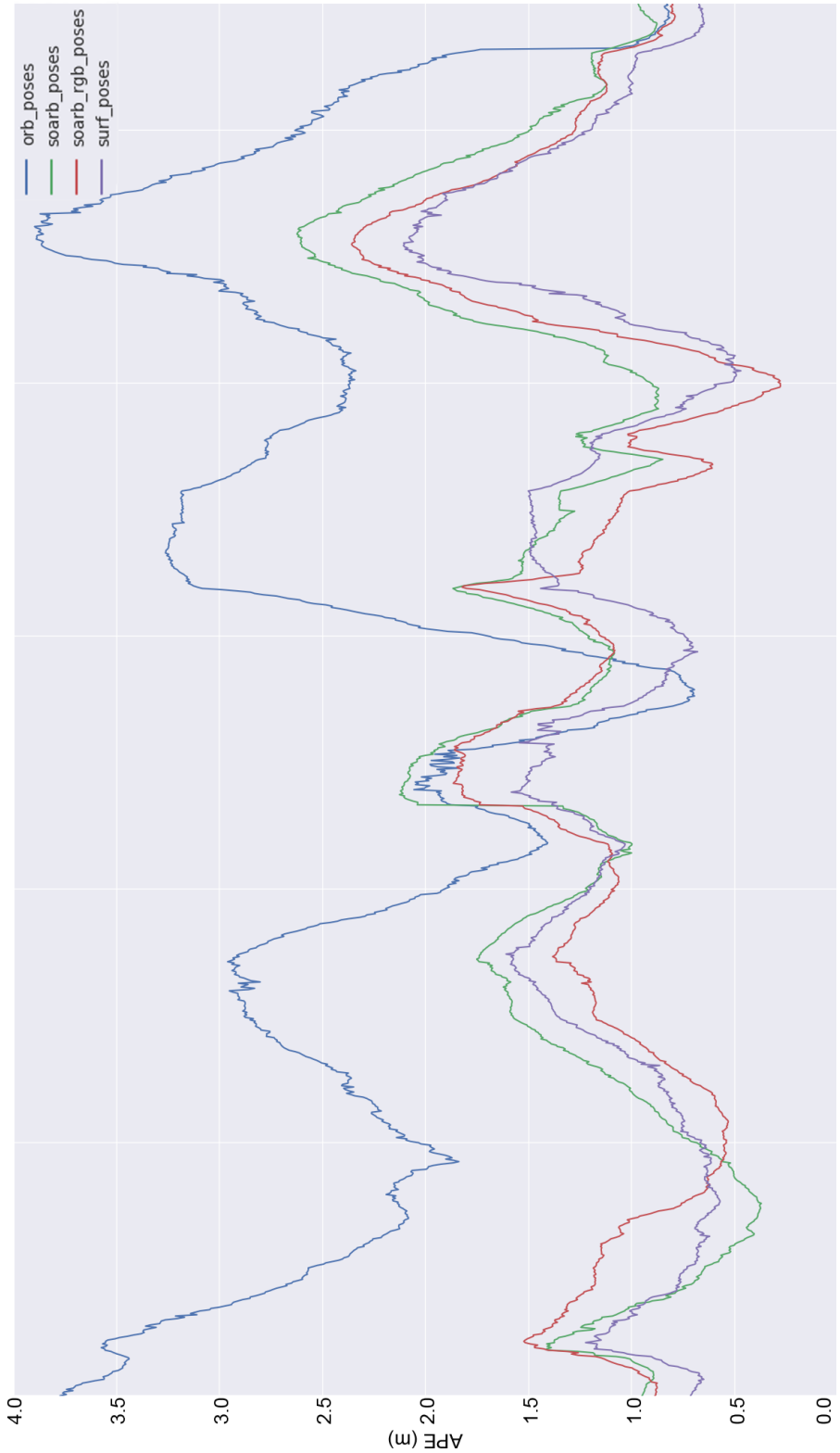


Abbildung 10.8: KITTI Dataset 07 - RTAB-Map Absoluter Positionsfehler (RBG-Datensatz / SOARB-RGB)

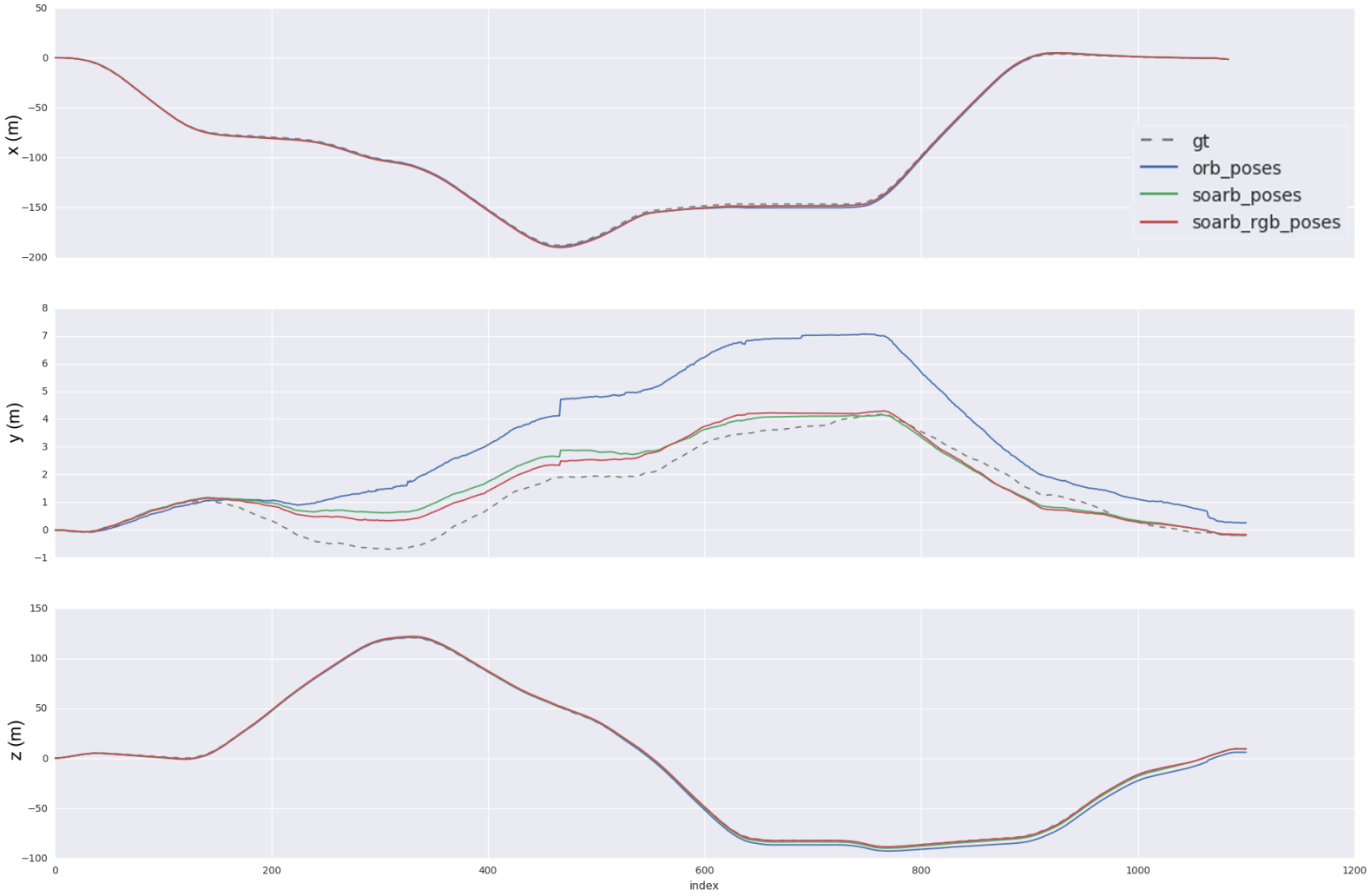


Abbildung 10.9: KITTI Dataset 07 - RTAB-Map Positionsverlauf in X-,Y- und Z-Richtung (RBG-Datensatz / SOARB-RGB)